Team For Capella

User Manual

Version v7.0.1, 2025-04-07

Contents

Chapter 1. Introduction to Team for Capella

Contents

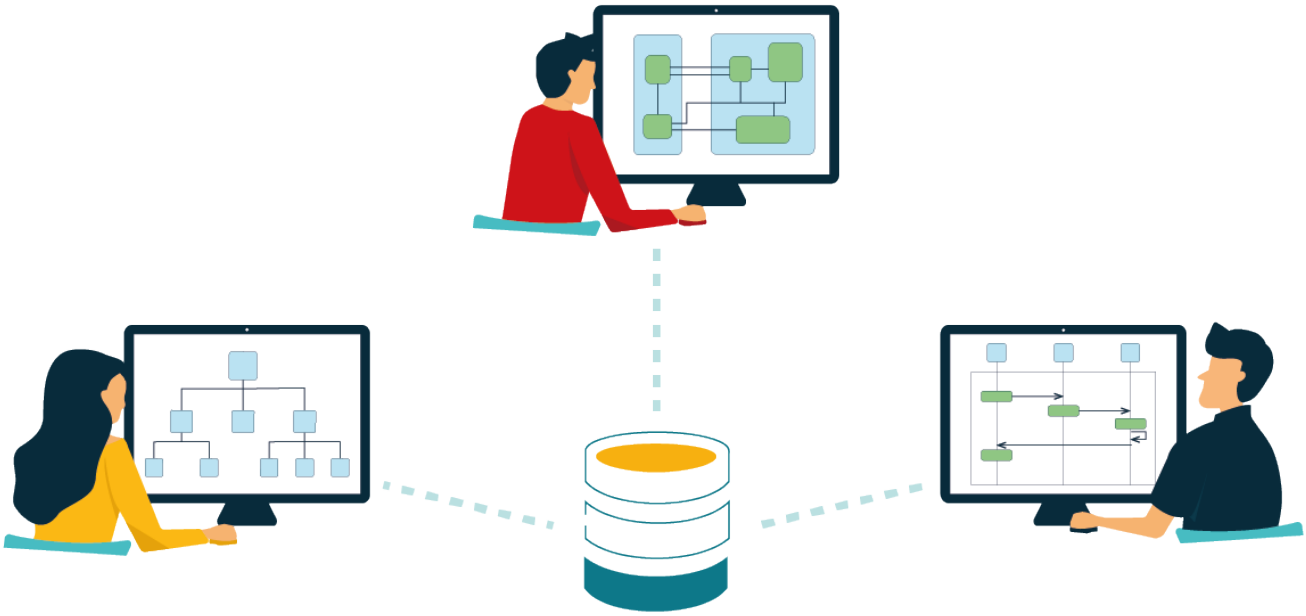
1.1. Overview

Team for Capella is an add-on that allows users to collaborate on remotely shared models and representations. For this collaboration between users to operate smoothly, Team for Capella relies on the following features:

- Simultaneous collaboration
  - Any object being edited is automatically locked and indicated to other users by a specific decorator. Only this object and its closest dependents are locked, allowing other users to continue working on the same model. These fine-grained locks are automatically released as soon as the modifications are saved. This allows several users working simultaneously on the same model.
- Instant updating
  - As soon as a modification on a model element is saved it is automatically and instantly propagated across all users' views. No need to manually refresh your model in order to retrieve modifications performed by other users: you are always working on up-to-date model elements.
- Explicit locking
  - When a user needs to work during a long period on the same set of model elements, he can explicitly lock these elements. The lock will only be released on-demand, as soon as the owner of the lock decides to allow other users to work on these elements
- Storage on a shared server
  - Team for Capella runs on a server shared across all your authorized team members. It can be administered to properly start and stop the system, and see who is currently connected. Models can be stored on one or several database(s) deployed on one or several machine(s).
- Sharing a local project
  - Modeling projects which are installed on your environment can be exported to the remote repository in order to be shared with other team members.
- Retrieving a remote project
  - Projects installed on the shared server can be manually imported into your environment or automatically saved to a backup server.
- Change history
  - History of commits is available to see which changes occurred on the shared models. At any

time, you can compare two versions to see the differences. You can also see all the model elements and diagrams impacted by several commits.

- Secured access
  - Data stored in the repositories can be protected by using LDAP to authenticate users, and by using SLL to encrypt the exchanges between the clients and the database(s). It is also possible to define access rights depending on user profiles.
- **Flexible licensing mode**
  - Our floating licensing mode allows you to deploy Team for Capella in a flexible way, depending on your context and your infrastructure: licenses are floating, allowing them to be shared among several users over time, when required due to low network's bandwidth, remote desktop mechanism is supported, avoiding you to deploy Team for Capella client on user's machines, large organizations working with Capella on several projects can deploy Team for Capella server on several machines simultaneously: the licensing mode only controls the number of current connected users, not the number of running servers.
- **Server administration with a web interface**
  - System administration and project lifecycle management does not require using Eclipse but is handle with a web interface. Indeed, Team for Capella installation can be completed with Jenkins used as a scheduler for various job managing the Capella project shared on a CDO server, such as automatic backups. The server used for sharing Capella project is also managed with Jenkins.



1.2. Roles Differentiation

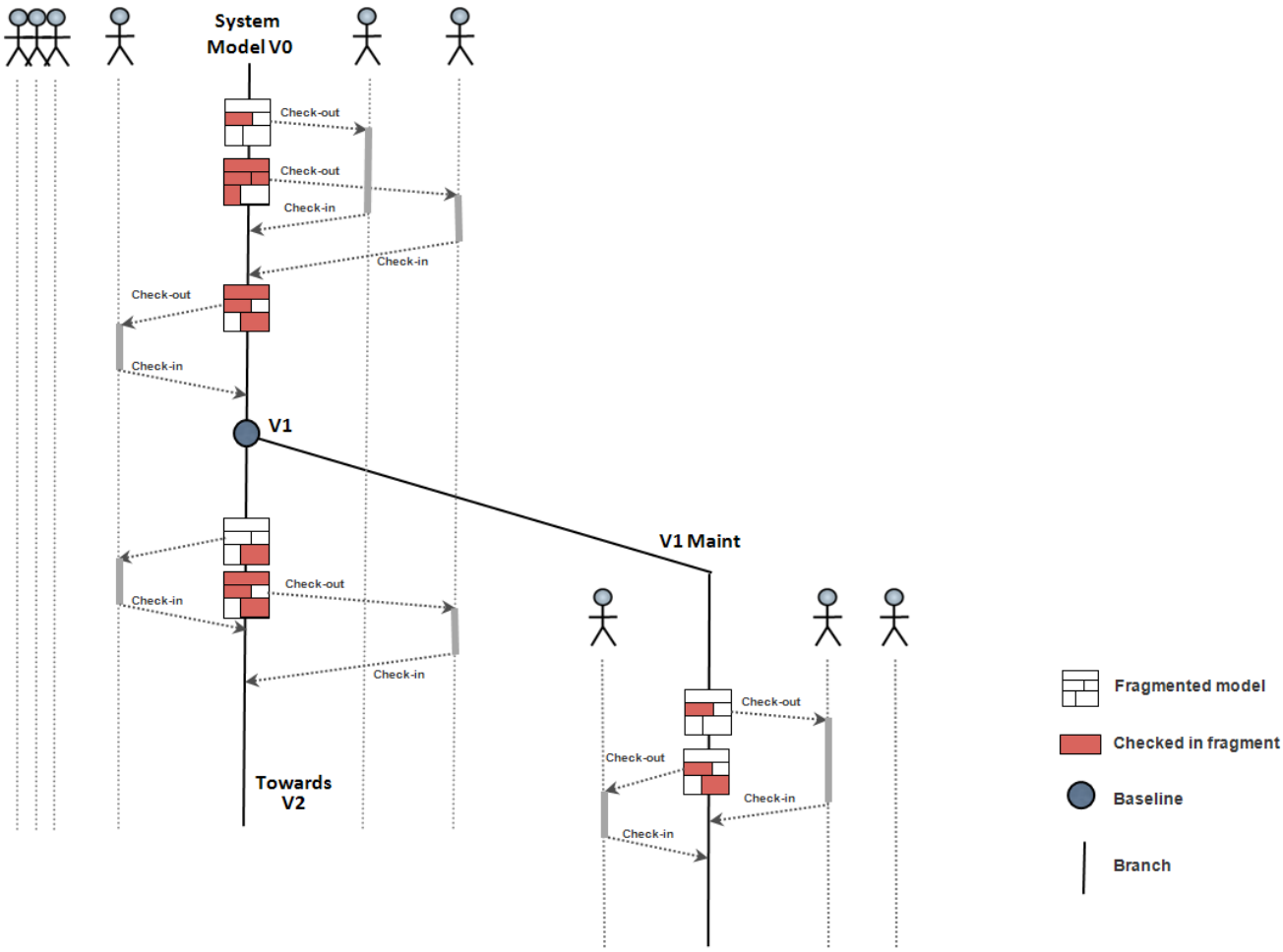
- This documentation is split between the different roles the team members can have while interacting with Team for Capella:
  - User : share and connect to remote Capella projects in order to work collaboratively with other users;
  - Project Administrator : manages the lifecycle of Capella projects and models;

- System Administrator : manages the server side of Team for Capella, such as installing the Jenkins scheduler, managing the CDO server and users' accesses;
- Developer : contribute new functionalities to extend Capella (or its underlying framework Sirius) thanks to its APIs.

1.3. Rationale and Concepts

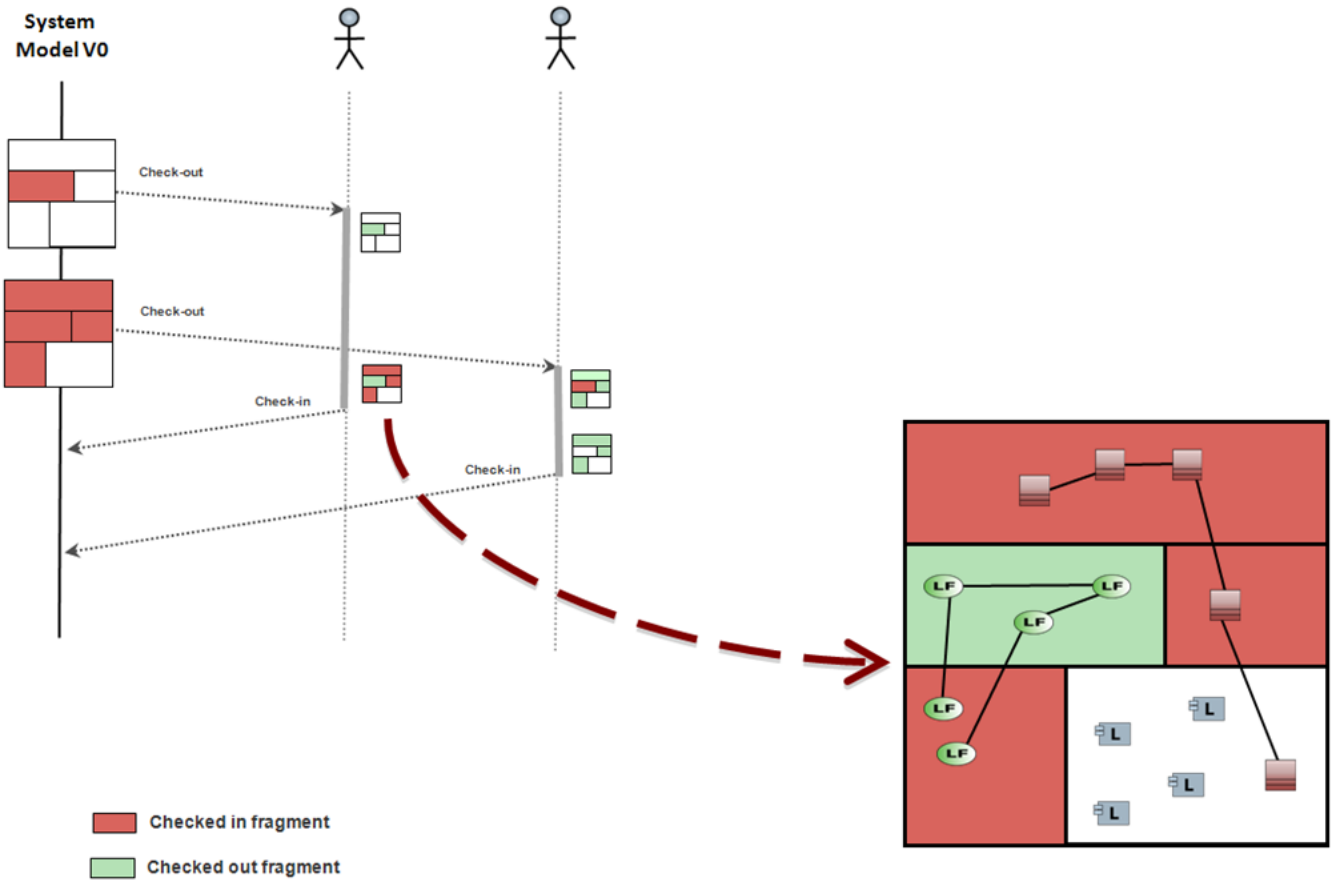
History: Collaborative Work Based on SCM Capabilities

- The primary role of a SCM tool is to manage versions of files (models, pieces of code, resources, etc...)
- A Model is split in files, called fragments
- To modify model elements belonging to one specific fragment, users must check out this particular fragment
- This fragment becomes read-only for the other users *(red)*
- A baseline *(V1)* is put on a regular basis when significant states of the model are reached
- Branches *(V1 Maint)* can be created starting from one specific baseline
- Diff Merge is useful to report changes from one branch to another, but is not used to manage concurrent accesses



- Models are graphs: elements are highly inter-connected (i.e. across fragments)

- Fragmentation is hierarchical
- This situation can rapidly lead to strong perturbations:
  - If fragments are too large, users will rapidly be stuck, waiting for particular fragments to be released
  - If fragments are too small, the additional non-functional tasks (check-out, check-in, etc.) becomes too heavy



Relying on a SCM tool to manage concurrent accesses is possible, but clearly limited.

This main reason is that the needs for managing model versions (genuine objective of a SCM tool) and concurrent accesses are deeply different:

- **Model versioning**: The need is to identify key intermediate baselines (for review, publication, validation, etc.), manage branches allowing maintaining several versions in parallel (development, maintenance, etc.), identity in which version a PCR is fixed, etc. Fragmentation of models should be limited to what has to be versioned.
- **Concurrent accesses**: The need is a granularity as fine as possible. From the end user point of view, the locking / unlocking mechanisms have to seamless (i.e. as transparent as possible) so that they do not interfere with their engineering activity. For example, there is often no need for associating each individual model modification to a UCM activity.

Here, fragments are created to manage concurrent accesses and not anymore

because their content has to be versioned.

The global idea of Team for Capella Solution is to separate the management of both needs:

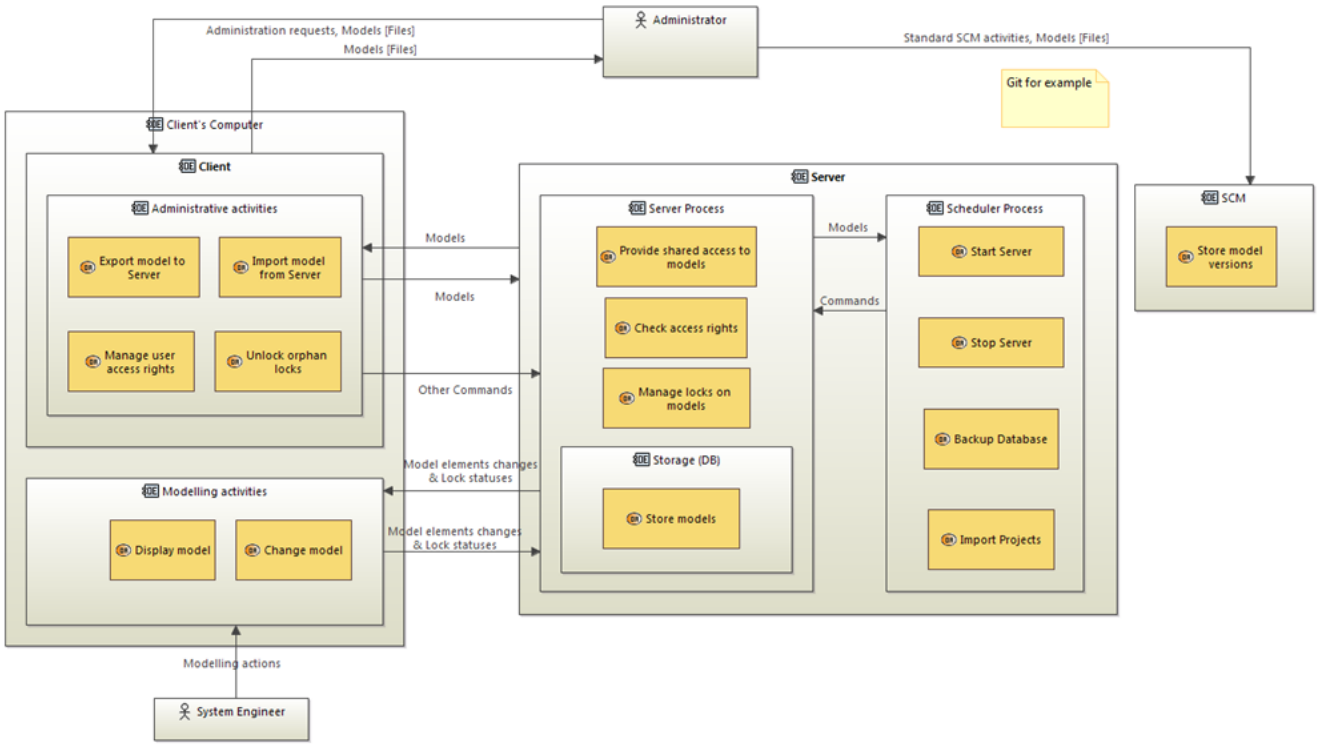
- **SCM tools are perfect for managing versions.**
- **Team for Capella solution only focuses on managing concurrent accesses.**

Team for Capella Solution

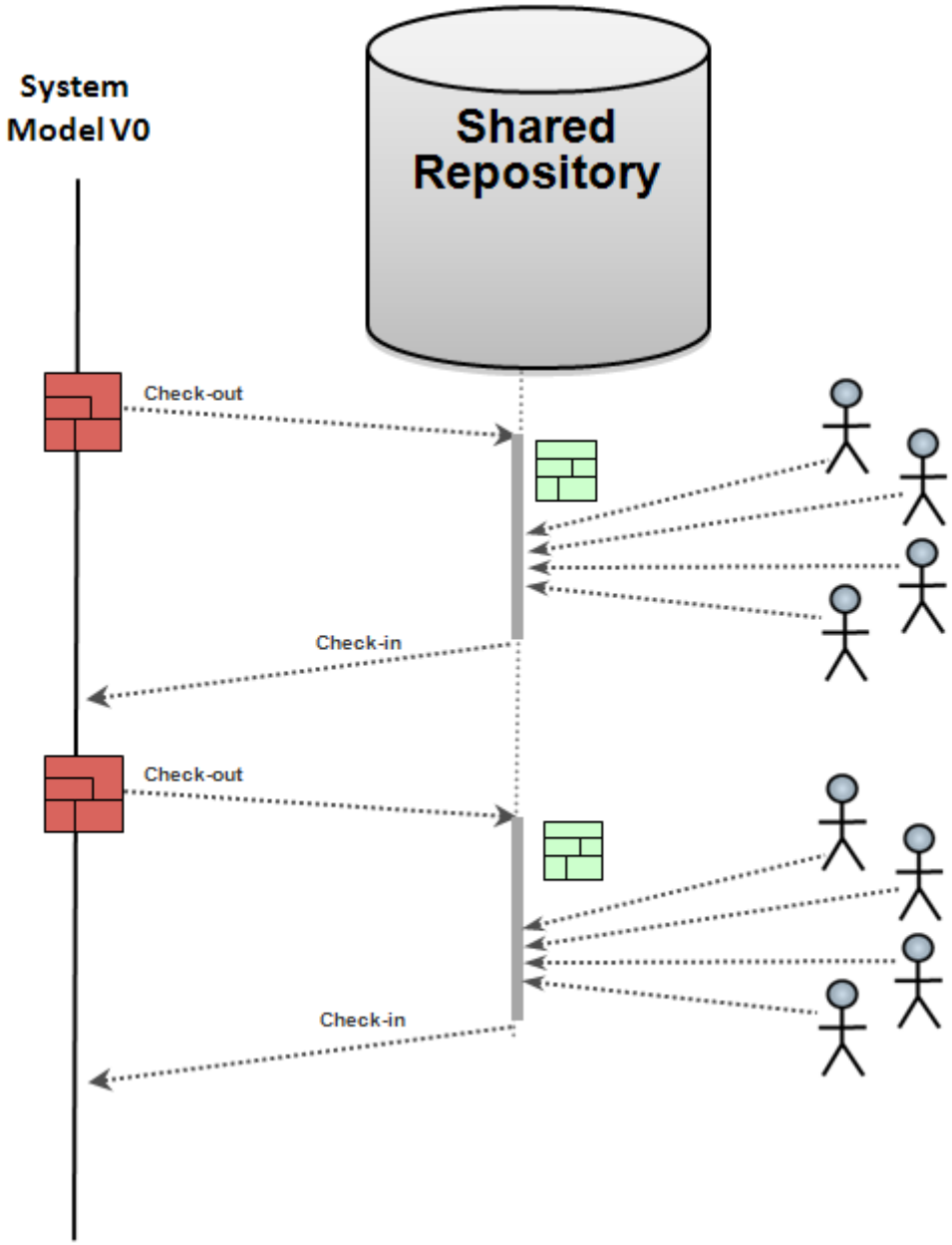
- **Main Ideas**
  - All users shall see an up to date version of the model.
  - Contributors shall be able to work on the same model without interfering, with a granularity as fine as model element.
  - Only one user shall cope with SCM concept.
- Main drivers:
  - **Use a Shared Repository**: File-based model is exported in a repository accessible by several users.
  - **Manage locks at model element granularity**: If one user needs to modify one single element, he should only lock this element.
  - **Make locks and update mechanisms automatic**: The locking / unlocking / update mechanisms have to be as transparent as possible so that these non-functional activities do not interfere with the engineering activity.

Team for Capella Solution: 3 products.

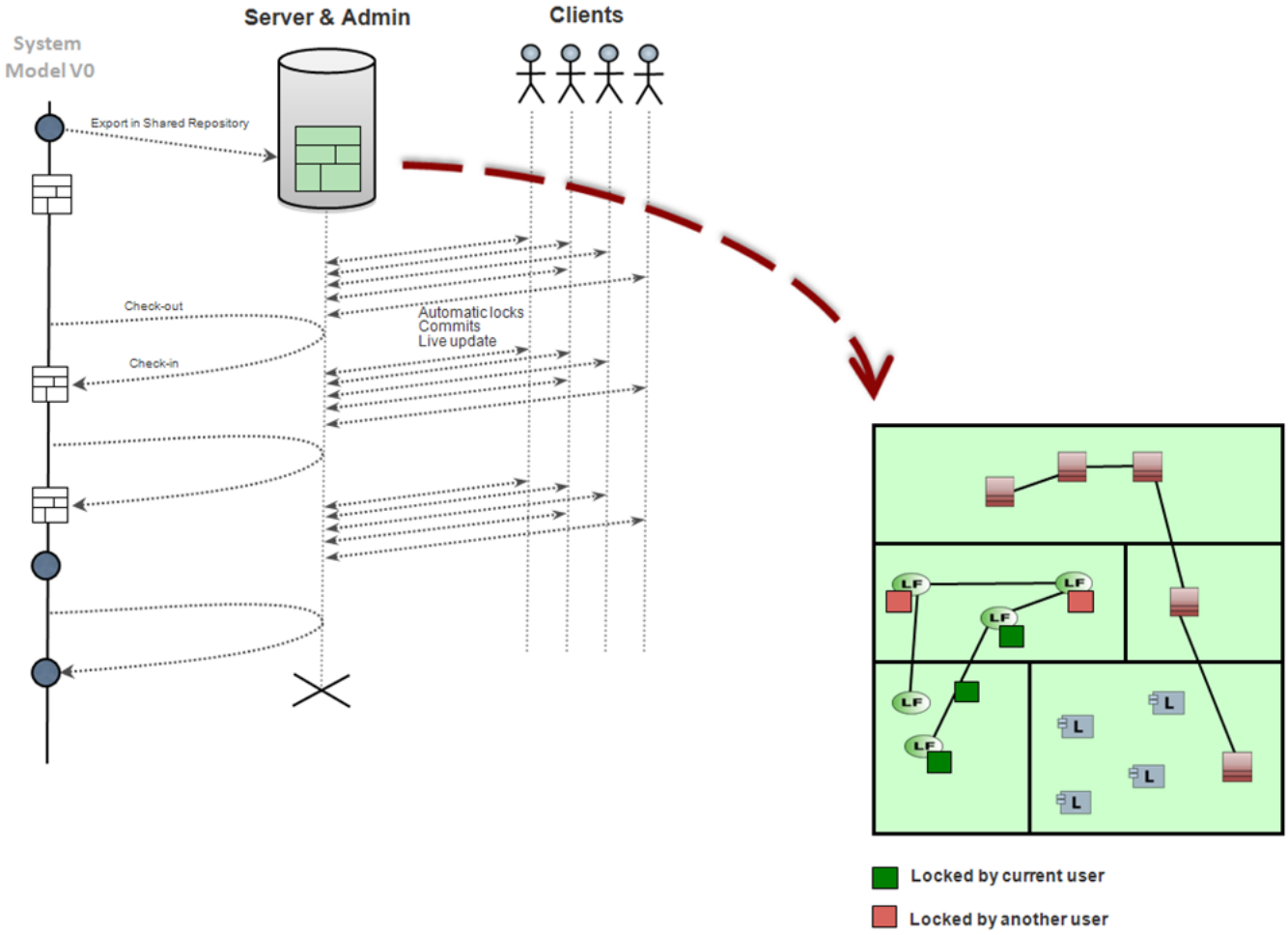
- **Team for Capella Client**: it is a standard Capella client with additional functionalities:
  - to work on a shared remote model,
  - to perform administrative tasks on the Team for Capella Server:
    - Import/Export a model from/to the Team for Capella Server,
    - Manage access rights,
    - Manage locks,
- **Team for Capella Server**: manages the repository, the locks and the access rights,
- **Team for Capella Scheduler**: a Jenkins server can be used to manage the Team for Capella Server:
  - Start/Stop the Team for Capella Server,
  - Do periodic imports of models and backups of the server's database.



- Only the **Team for Capella Admininistrator** has to work with the SCM tool,
  - He/She has to regularly push back shared models to Git,
  - He/She has to created Baselines when necessary,

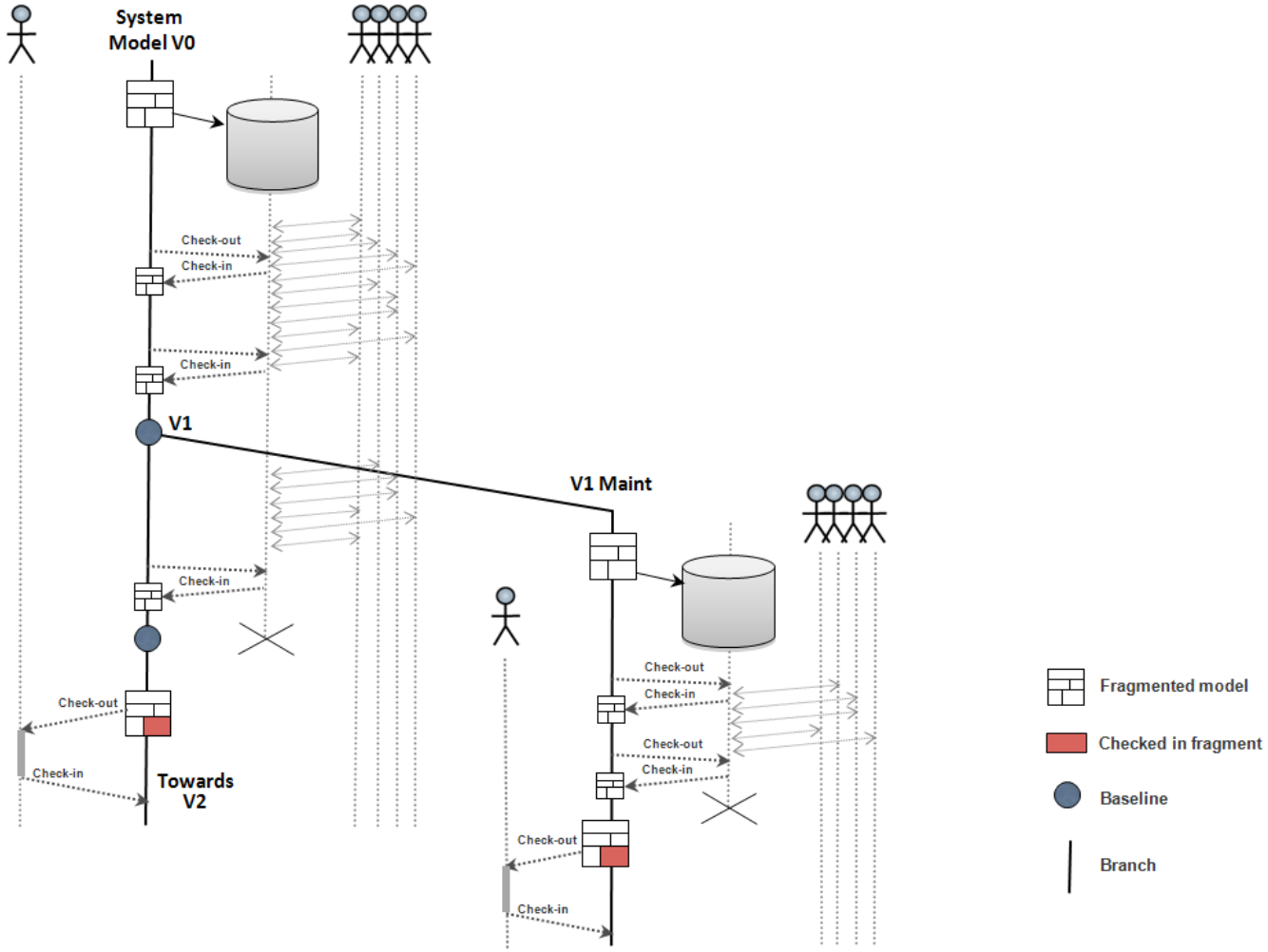


- The **Team for Capella Server** is responsible for managing fine-grained concurrent accesses to the model,
- Users connect to the shared repository through **Team for Capella Client**,
  - Team for Capella Client is connected in live mode: it always shows the latest state of the model shared in the repository,
- Fragmentation is only used for model versioning purpose.



Shared Repositories and Configuration Management

- **Shared repositories are not permanent**
  - They live as long as concurrent accesses are necessary on one version of a model
- It is still possible for users to contribute directly through Git once the repository has been shut down
- Diff Merge is useful to report changes from one branch to another, but is not used to manage concurrent accesses


System
Model V0
Check-out
Check-in
Check-in
V1
V1 Maint
Check-in
Check-out
Check-in
Towards
V2
Check-out
Check-in
Check-out
Check-in
Check-out
Check-in
Fragmented model
Checked in fragment
Baseline
Branch

Chapter 2. Release Note

Contents

The Team for Capella 7.0.1 release notes are available at:

https://www.obeo.fr/en/team-for-capella-releases#7.0.1

2.1. What's new and API changes

The release note is updated for each new version and contains descriptions on changes visible by users, new or modified APIs accessible for developers. The change log can also be found online: Team for Capella Change Log

Changes in Team for Capella 7.0.1 (from 7.0.0)

Compatibility with Capella 7.0.1

UX enhancements

ADDED: The *Semantic Criteria* category has been added in the *Filters and Customization* dialog opened from the Diff/Merge editor when comparing two models with *Compare with* > *Compare Each others*.

ADDED: The Semantic Browser now retrieves the impacted diagrams for newly created semantic objects, ie. the model elements created in a connected project which has not been saved since their creation (those objects are not known yet for the server nor for the other users).

MODIFIED: The Semantic Browser refresh computation of referenced elements has been reworked for elements of connected projects. Computation is now done in less than 1s for some projects which used to encounter a 1min freeze on selection change.

MODIFIED: The refresh of frozen diagram, ie. diagrams which were *locked by other* before reception of remote changes, has been reworked to avoid interfaces freezes occurring in some situations on xAB, xDFB diagrams having functional chains, physical paths or physical links.

Packaging, installation and deployment

MODIFIED: The default repository's data source is now named *repoCapella* instead of *capella* to match its repository name, ensure naming consistency and ease readability.

MODIFIED: In multi-user configuration with shared Capella installation folder, it is now possible to keep the registered repositories list in the shared *capella/configuration* folder.

Scheduler

ADDED: The *Database - Restore* job is now able to restore dynamic repositories and/or a single repository at a time.

ADDED: A new option has been added to the *Repository - Start* job to allow the re-initalization of a repository without having to restart the server nor to access the files.

MODIFIED: URLs of Jenkins plugins have been replaced in installation scripts in order to avoid redirect errors during installation on Linux.

Server

ADDED: A new option has been added to the *Start repository* REST API to allow the re-initalization of a repository without having to restart the server nor to access the files.

ADDED: Two new OSGI commands *revisionCacheInfo* and *revisionCacheCleanup* are available to manage the internal revisions cache. They can be called on the OSGI console or from the REST API.

ADDED: The OpenID Connect authentication is now officially compatible with Keycloak.

ADDED: Loggers have been added to log actions that are processed such as users opening/closing a session, committing changes or actions on repositories through the REST API (creating/deleting users/projects/repositories...).

MODIFIED: The OpenID Connect authenticator has been reworked to improve the token validation and allow the customization of the authentication request sent to the identity provider and the user registration capabilities.

MODIFIED: Embedded Swagger UI has been updated from 5.11.2 (2024-01-29) to 5.20.1 (2025-03-10).

Tools

ADDED: A new CLI application is available to create a connected projects. *connect.bat/connect.sh* scripts have been added to the *TeamForCapella/tools* folder.

Notable fixes

MODIFIED: Import wizard now correctly sets the Library nature if the imported project is a Library.

MODIFIED: Export wizard now better handles several use cases mixing multi-selection, project names containing whitespaces, libraries. Better guards and error messages have been added to report Merge Strategy known unsupported cases.

MODIFIED: Import/export wizard usage no more forbid later activation of durable locks for a given Capella workspace.

MODIFIED: The activation of the *Next* button of Import/Export/Connect wizards is now based on the result of the *Test Connection* button.

MODIFIED: OpendID Connect logout and authentication cancel no more lead to project opening.

MODIFIED: *License Server - Run* job console is no more spammed by Jetty debug logs.

MODIFIED: Exporter/Importer/Connect/Credentials CLI application better handle the *-help*, *httpHost*, *-connectionType* arguments.

API Changes

Changes in plugin `com.thalesgroup.mde.cdo.emf.transaction`

MODIFIED: `com.thalesgroup.mde.melody.collab.MelodyCDORepositoryManager` and `com.thalesgroup.mde.melody.collab.query.CDOAirdResourceQuery` have been moved from plugin `com.thalesgroup.mde.melody.collab.ui`.

Changes in plugin `com.thalesgroup.mde.melody.collab.ui`

MODIFIED: `com.thalesgroup.mde.melody.collab.ui.navigator.wizard.MelodyConnectToRemoteWizard` `fr.obeo.dsl.viewpoint.collab.ui.api.wizards.ConnectToRemoteWizard`

REMOVED: `com.thalesgroup.mde.melody.collab.MelodyCDORepositoryManager` and `com.thalesgroup.mde.melody.collab.query.CDOAirdResourceQuery.CDOAirdResourceQuery` have been move to plugin `com.thalesgroup.mde.cdo.emf.transaction`.

Changes in plugin `com.thalesgroup.mde.melody.team.compare`

MODIFIED: `com.thalesgroup.mde.melody.team.compare.MelodyTeamDifferenceCategoryProvider` now inherits from `org.polarsys.capella.core.compare.CapellaDifferenceCategoryProvider` instead of `org.eclipse.emf.diffmerge.ui.sirius.SiriusDifferenceCategoryProvider` in order to get the Capella category all filters from the Capella project explorer.

Changes in Team for Capella 7.0.0 (from 6.1.0)

Compatibility with Capella 7.0.0

UX enhancements

ADDED: The Commit History view now allows to filter the displayed impacted elements.

MODIFIED: Connection wizards now select the new project at the end

MODIFIED: Export commit history actions have been modified to propose to squash consecutive commits done by the same user with the same description (same mode available in the Commit History view and the Importer CLI)

Packaging, installation and deployment

ADDED: Compatibility with Capella 7.0.0.

Internal components update to match the same platform: Eclipse 2023-03, Sirius 7.4.1, Collaborative layer for Sirius 15.3.0, Jetty 10.0.20 and CDO 4.22 (forked by Obeo).

ADDED: Installation guide has been completed with two additions:

- Jenkins offline installation and configuration procedure
- Advanced installation and configuration sample procedure

ADDED: Dedicated client only installation bundles are available for Windows, Linux and macOS.

ADDED: Obeo license system and server have been modernized : moved to the same components than Team for Capella (Jetty, Eclipse, Java, ...). It is now possible to use a HTTP(s) transport to exchange tokens and not only the Socket based one.

Scheduler

ADDED: New jobs have been added in Jenkins :

- Repository - Commit history: it allows to get the commit history of a repository.
- Repository - List projects: it allows to get the list of projects on a repository at specific timestamp.
- Repository - Import projects from history: it allows to import projects from a repository from a specific timestamp.

MODIFIED: The targeted Jenkins version has been updated from LTS 2.375.3 to LTS version 2.440.3.

MODIFIED: Jenkins/Scheduler installation scripts have been improved: parameters/variables have been extracted to a sibling .properties file.

MODIFIED: Modification of some default jobs to extract the repository name to a combo choice.

MODIFIED: The jobs status have been modified to be consistent with the executed command status.

MODIFIED: The list of recommended Jenkins plugins has been reduced.

Server

ADDED: com.thalesgroup.mde.melody.db.h2.H2Adapter has been updated to have the commit description consider has a CLOB in the database instead of a VARCHAR. This now allows to commit long text as a commit description. However, setting the property *fr.obeo.dsl.viewpoint.collab.common.commit.description.max.length* to an Integer value below or equal to 255 will automatically revert the commit description DBfield to a VARCHAR. Changing this property requires to reinitialize the repository.

ADDED: A new server monitoring capability has been added through 3 REST endpoints: `/monitoring/metrics`, `/monitoring/info` and `/monitoring/health`.

ADDED: It is now possible to store the default credentials of several servers in the secure storage.

MODIFIED: REST API is now able to better manage static repositories

- ability to handle static repositories in users services (list, add, delete, update)
- correctly manage static repositories in start/stop repository endpoints
- addition of the authentication type info for static repo in list repositories endpoint result

MODIFIED: Internal database backup process logs have been improved to be less confusing.

Tools

ADDED: T4C tools now handle the -eclipse.password option and can target a dedicated secure storage on Linux.

MODIFIED: Importer application arguments have been modified to be able to export a specific version from a repository

MODIFIED: Timestamps arguments and format in results files have been homogenized

MODIFIED: Importer error message when the repository is stopped has been improved

REMOVED: The -archiveFolder argument has been *removed*. It used to the folder where to zip projects. The use of -outputFolder must now be required (and -archiveProject=true but true is its default value).

REMOVED: Telnet support has been removed. It was deprecated since 6.1.

Notable fixes

MODIFIED: OpenID component now correctly handles redirect URLs in its responses when placed behind an HTTPs proxy

MODIFIED: Files are no more flattens in projects imported by the CLI importer application

MODIFIED: Diagnostic job can now target dynamic repositories and/or server with WS/WSS connection types.

MODIFIED: Asynchronous behavior of some GMF actions in diagram toolbars is now better handled in order to avoid potential deadlock on remote changes receptions.

MODIFIED: Behavior of server shutdown and repository stop detection have been improved to avoid potential deadlocks. The effective close behavior has not changed: project sessions and opened editors are closed, unsaved data is lost.

MODIFIED: A few other minor fixes around collaborative features or the compatibility between Capella features and the collaborative add-on.

API Changes

Changes in plugin `com.thalesgroup.mde.melody.collab.command`

MODIFIED: `com.thalesgroup.mde.melody.collab.common.SemanticResourcesHelper.getAllReferencedSemanticResources(CDOView, URI)` has been changed from `org.eclipse.emf.cdo.transaction.CDOTransaction` to its super interface `org.eclipse.emf.cdo.view.CDOView`.

Changes in plugin `com.thalesgroup.mde.melody.collab.command.console`

Plugin `com.thalesgroup.mde.melody.collab.command.console` has been removed. It used to provide

capability to send command to the Team for Capella server application via Telnet. Its API classes are no longer available: `com.thalesgroup.mde.melody.collab.client.Activator`, `com.thalesgroup.mde.melody.collab.client.Application` and `com.thalesgroup.mde.melody.collab.client.Main`.

Changes in plugin `com.thalesgroup.mde.melody.collab.importer`

REMOVED: `CONSOLE_PORT` and `CONSOLE_TIMEOUT` unused constants have been removed from `com.thalesgroup.mde.melody.collab.importer.api.TeamServerConnectionsConstants`. They used to be Importer/EXporter CLI parameter related to the removed Telnet support.

Changes in Team for Capella 6.1.0 (from 6.0.0)

Compatibility with Capella 6.1.0

UX enhancement

ADDED: A new option is available in *Import model from remote repository* wizard to configure the override strategy when the imported project already exists in the workspace or on the file system. See details in User Guide.

MODIFIED: Libraries can now be managed on a connected project : references to shared libraries can now be added or removed.

MODIFIED: The *Merge* strategy of the export wizard has been improved to handle the representations (diagram and tables) and their content. The identity and commit history is now also kept for those elements. Additional fixes have been done to minimize differences and avoid to do an export commit when there is no change to send to the server.

MODIFIED: The *Commit History* view no more displays the changes of wizards and technical commits. The number of new/modified/deleted objects is displays instead. Contrary to commits done by the save actions, those commits have a description which is automatically filled by the client and which contains the impacted project or resource name and begins with one of the following tags: [Export], [Delete], [Maintenance], [User Profile], [Import], [Dump].

MODIFIED: Remote images used in diagrams and Rich Text descriptions can now be displayed by other users.

MODIFIED: After the use of the *Dump* wizard, a manual refresh of the project in the *Project Explorer* is no more required to see the all the result files.

MODIFIED: Frozen diagrams no more react to reception of remote connection bendpoints changes.

Packaging, installation and deployment

MODIFIED: The targeted Jenkins version has been updated from LTS version 2.332.3 to LTS version 2.375.3. This brings several important security fixes and also notable changes as documented in the LTS release notes.

MODIFIED: The Linux bundle Scheduler jobs have been improved to avoid permissions issues.

MODIFIED: The *Project - Automatic Import and push to Git - Template* Windows and Linux job templates have been improved to work on a branch with no CommitHistory.txt file.

Server

ADDED: It is now possible to start/stop a single repository (either static or dynamic) without having to start/stop the whole server. A *Server - Start repository* job has been added in default Scheduler jobs.

MODIFIED: The Server packaging has been reworked to keep only necessary and non-ui plugins.

Tools

ADDED: The timezone *-from* and *-to* parameters of the Importer application are now able to handle the timezone in the given timestamps.

ADDED: It is now possible to compute impacted representation for each commit with the use of *-computeImpactedRepresentationsForCommitHistoryChanges true* when changes are included in the commit history extraction (*-includeCommitHistoryChanges true*) done by the Importer application.

MODIFIED: In order to improve performances and reduce noise, the commit history changes are no more extracted for commits which corresponds wizards and technical commits.

MODIFIED: The Export application *Merge* strategy has been improved to better configure Diff/Merge and have a consistent behavior compared to the *Export model to remote repository* wizard.

Changes in Team for Capella 6.0.0 (from 5.2.0)

Compatibility with Capella 6.0.0

UX enhancement

ADDED: A new dialog has been added to ease the selection of images when setting a style to workspace image to a node. The display of this dialog allows to access the images of the repository in "Gallery" mode to have a preview of the available images. The dialog also provides features to modify, delete or add new images to the repository.

ADDED: Images on server can also be managed via the pop-up menu on the "*.aird" file or on the shared project > Manage Images from Remote Server.

ADDED: A dialog that displays the result of the import/export process has been added at the end of the import/export wizards.

ADDED: A new override strategy is available in export wizard: the *Merge* strategy allows to use Diff/Merge to compare local project and existing remote project in order to commit only computed deltas, keep model elements identity and commit history.

ADDED: Wizard and technical commits done outside the Save action now have a dedicated commit description allowing to identify them in the Commit History view.

ADDED: Several actions and views provided by underlying components but not related to Team for Capella user experience have been hidden thanks to dedicated capabilities disabled by default.

MODIFIED: Team for Capella wizards projects creation/import/export have been renamed:

- *New > Connect to remote model* wizard has been replaced by *New > Capella Connected Project*
- *Export... / Team for Capella / Export model to remote repository* wizard has been replaced by *Export... / Team for Capella / Capella Project to Remote Repository*
- *Import... / Team for Capella / Import model from remote repository* wizard has been replaced by *Import... / Team for Capella / Capella Project from Remote Repository*

MODIFIED: WS and WSS connection types are now available alongside TCP and SSL in Repository configuration page and wizard pages.

Packaging, installation and deployment

ADDED: The Team for Capella client-side administration features are no longer installed by default. A dedicated installation script is provided in *tools/* folder. Then the corresponding capabilities need to be enabled.

MODIFIED: The targeted Jenkins version has been updated from LTS version 2.303.3 to LTS version 2.332.3. This brings several important security fixes and also notable changes as documented in the LTS release notes.

MODIFIED: Until 6.0, Telnet was used so that send OSGI commands from tools applications (importer, maintenance, console) to the server application. In 6.0, **Telnet commands are now deprecated** and have been replaced by calls to the REST Admin API.

MODIFIED: The web socket and web socket secured protocols (ws:// and wss://) are no longer flagged as experimental.

MODIFIED: Jenkins jobs have been updated to use applications with the admin server configuration.

MODIFIED: The linux bundle is no longer flagged as experimental for both Team For Capella client features and server.

Server changes

ADDED: The REST Admin Server feature is no longer flagged as experimental. It is now installed and enabled by default. It is automatically started when starting the Team for Capella Server. See Server administration and Server configuration pages.

ADDED: In replacement to the console application, that was used to call service on the server via Telnet, the HTTP Requester application has been added to call service on the server using the jetty admin server.

MODIFIED: Some parameters of importer, exporter, maintenance applications have been updated. Some have been renamed and some added such as httpXXX parameters used to request the jetty admin server.

MODIFIED: The Team for Capella server product does not contain *UI* plugins from Sirius, Capella, CDO nor Eclipse platform anymore. When launched without the server-side client tools (Importer, Exporter, Maintenance), it can now be launched without a X Server.

Tools

ADDED: `-archiveCdoExportResult` argument has been added in order to zip (or not) the XML file resulting from the the cdo export command launched by the importer in intermediate step. When the XML file is zipped, the zip is created into the "output folder" (see arguments of the T4C importer) and the original XML file is then deleted. The default value is true.

ADDED: Addition of the **Exporter** application. This application allows to trigger the export of local given projects with representation on CDO repository. More informations about this application can be found on Exporter strategy documentation.

ADDED: `-stopRepositoryOnFailure` argument has been added in order to stop the repository when import/export is on failure. This parameter could not be set to true if `-closeServerOnFailure` argument is already set to true.

MODIFIED: The Importer constant `com.thalesgroup.mde.melody.collab.importer.api.TeamImporterConstants` used especially for telnet command have been reported in a new class `com.thalesgroup.mde.melody.collab.importer.api.TeamServerConnectionsConstants` in order to share arguments between importer and Exporter application.

MODIFIED: In order to share arguments between importer and Exporter application, following arguments `-importerLogin`, `-importerPassword` and `-importerCredentials` have been renamed into `-repositoryLogin`, `-repositoryPassword` and `-repositoryCredentials`.

Changes in Team for Capella 5.2.0 (from 5.1.0)

Compatibility with Capella 5.2.0

UX enhancement

MODIFIED: In the Export project wizard, the behavior of the *Override existing resources* has been improved: whereas on previous versions the override was simply forbidden as soon as a Library project was detected on the server, now the export is forbidden if one of the resources to override is a Library resource and at least one of the remote projects is not overriden.

MODIFIED: Location selection page displayed at representation creation (and move) has been improved to use less ambiguous labels and terms for end-users.

Locks management

MODIFIED: The save action now sends the commit data and the unlock messages in the same network request. This allows to optimize the network usage and also to minimize the number of notifications to sent to all connected users and the implied distribution potential issues.

MODIFIED: All impacted elements by a deletion are now locked as soon as possible before

executing the delete command. After a deletion (ie after having validate the impact analysis dialog), some impacted semantic elements by the deletion (such as the container of a deleted element) were locked only in a second time. Most of the time, that was not visible by the end user but a second lock message was later sent to the server. That might lead to interrupt the delete during its execution if a lock instance exception was received.

Packaging, installation and deployment

REMOVED: Jenkins has been removed from Team for Capella bundles. Jenkins becomes a required installed software to deploy the Scheduler.

ADDED: The bundle now contains documentation and scripts to help the installation and configuration of Jenkins.

- The provided scripts and configuration tips allow to retrieve the same set of Jobs, Views of the Scheduler as in Team for Capella 5.1.0.
- The installation and configuration process and script have been described and tested with Jenkins LTS 2.303.3.
- The System Administration Guide have been complete with Jenkins installation page.

MODIFIED: The targeted Jenkins version has been updated from LTS version 2.277.3 to LTS version 2.303.3. This brings several important security fixes and also notable changes as documented in the LTS release notes.

Server

ADDED: OpenId Connect authentication support has been added to Team for Capella. The configuration to use OpenID Connect targeting MS Azure AD can be retrieved in the Activate OpenID Connect authentication documentation section.

Tools

ADDED: New API has been added to send timeout specification to the OSGI console client. The parameter *-consoleTimeout* can now be used for the importer and maintenance applications and jobs to define the timeout for the commands sent to the server. If a command is stuck too long compared to the timeout, a SocketTimeoutException will be thrown.

MODIFIED: The importer application has received several improvements

- A lock is acquired on aird resources before performing the CDO XML dump. This action prevents other connected users to commit their changes during the XML dump. This modification concerns only the *Snapshot import* importer strategy. (See Importer strategies for more details).
- The import steps order has changed to avoid issues with imported images. There is now a first step with import of projects and a second with archiving of projects. The number of projects in "Import to local final status" can be different of the number in "Archiving final status". Indeed, some dependency projects can also be archived.
- The parameter *-includeCommitHistoryChanges* now works with *Snapshot* and *Offline* import

strategies.

- Images from Libraries are now correctly imported by the Importer application.
- It is now possible to import a project whose Project element status has been set.

Changes in Team for Capella 5.1.0 (from 5.0.0)

Compatibility with Capella 5.1.0

UX enhancement

ADDED: New shortcuts to Team for Capella wizards have been added to *File* > *New* menu.

ADDED: It is now possible to configure some preferences at the project level: this is the case for two preferences for Sirius: *Automatic refresh* and *Refresh at opening*.

- Local Capella projects now have the same behavior in a Team for Capella client than in Capella: *Automatic refresh* and *Refresh at opening* are enabled by default at the workspace level.
- However, shared projects have the same behavior as in Team for Capella previous versions: *Automatic refresh* and *Refresh at opening* are disabled by default at the project level. This can be modified in the connection wizard or later in the project properties. Refer to Overriding Sirius refresh preferences for a particular connected project.

ADDED: Two new property pages have been added in the properties of *.aird* files (available from the Properties contextual menu):

- Sirius Session Details : displays information about session resources, viewpoints, representations (number, load state). Invalid representations or representations which seems to need a manual refresh are listed.
- Collaborative Session Details (previously named *Repository Information*): displays repository connection information, login of connected users and locks (implicit and explicit locks taken by the current users and locks from other users).

MODIFIED: A diagram is now locked when changing its contextual elements

MODIFIED: Team for Capella wizard now follow the same rules than Capella and forbids project creation with names containing special characters.

Scheduler

MODIFIED: The Jenkins version has been updated to from LTS version 2.204.6 (LTS) to version 2.277.3 (LTS). This brings several important security fixes and also notable changes as documented in the LTS release notes.

MODIFIED: On each tab (except all because it's a derived tab), the column with the build button has been moved between the weather column and the project name column. It is more user friendly this way.

MODIFIED: Windows Service installation is now possible when Team for Capella installation path contains whitespaces.

REMOVED: CVS and SVN plugins have been removed from the Scheduler.

Server

MODIFIED: The default configuration of the CDO repository has been changed from *Auditing with Ranges* to *Auditing*.

- This changes the way to store lists in the internal database and improves server serialization and read performances with noticeable improvements on the user side in Semantic Browser and Commit history refresh.

Tools

Importer

MODIFIED: Management of images used in diagrams has been improved:

- When uploading an image, the user can choose if the image is embedded in the project or considered as an external image.
- When importing a remote project, remote images are properly copied: embedded images are copied within the imported project and external images are copied in the project they initially belonged to.
- All imported projects (Capella, library or projets containing images) are properly zipped by the importer job.

MODIFIED: A bug causing a blank result for the import of the commit history has been resolved.

ADDED: Session details are now logged during session check step (when *-checkSession* parameter is set to true).

Maintenance

ADDED: New cases of inconsistency are detected by the diagnostic job and can be repaired by the maintenance job: references in the model linked to missing elements in the database.

Experimental

As ***experimental*** features:

ADDED: Team For Capella client and server support web socket and web socket secured protocol (ws:// and wss://)

ADDED: A linux bundle is available for both Team For Capella client features and server.

Changes in Team for Capella 5.0.0 (from 1.4.2)

Compatibility with Capella 5.0.0

UX enhancements

ADDED: It is now possible to register several Team for Capella repositories in the Team for Capella client.

ADDED: A new Invalid representations is now displayed in the Project Explorer to easily retrieve invalid representations of a shared project.

ADDED: The Commit History view now allows to filter the displayed impacted elements.

MODIFIED: The wizard pages which allows to choose the location of created/moved representations have been improved for a better management and understanding.

Packaging, installation and deployment

The packaging has been reworked to ease installation:

MODIFIED: installation scripts and tools have been moved to a tools folder,

ADDED: plugin customization file is created during installation with default values,

ADDED: The server, license server and tools use the JVM provided by the Capella 5.0.0 bundle (OpenJDK 14.0.2).

ADDED: The Scheduler uses its own JVM as runtime (Jenkins is not yet compatible with Java 14, an AdoptOpenJDK 8u265 is embedded in the Scheduler.

MODIFIED: The logs of the server, license server and tools used from the Scheduler jobs are now directly visible in the Scheduler console view.

Changes in Team for Capella 1.4.2 (from 1.4.1)

- Compatibility with Capella 1.4.2
- Performances regressions on Collaborative features have been solved in Capella 1.4.2
  - The lazy loading of representations in Team for Capella 1.4.1 was broken by the first creation of a new diagram by a user.
    - The regression was introduced in 1.4.1 with the new TitleBlock feature even if not used.
  - Some operations might create unwanted Operational Analysis Entities as a silent side effect (Bug 566264).
    - The issue used to occur on OAB diagrams with contextual Operational process allocated to a hierarchy of entities.
    - The Entity elements wrongly generated on each call of the problematic method can cause performance issues. Those elements can be retrieved from the Capella Project Explorer and manually deleted.
- The Maintenance application is now compatible with User Profile mode

Changes in Team for Capella 1.4.1 (from 1.4.0)

Change Management

ADDED: It is now possible to cancel the save from the commit description dialog. When clicking on "Cancel", the user changes are kept unsaved locally and no commit is performed.

Scheduler

MODIFIED: The Jenkins version has been updated to from LTS version 2.150.2 (LTS) to version 2.204.6 (LTS).

ADDED: Two new jobs have added as Jenkins jobs: *Restore backup* which is the twin of the existing *Backup database* job and *List connected projects and locks on model*. Refer to Jenkins configuration

MODIFIED: The scheduler jobs are organized in categories that are called views in Jenkins: *Server Management*, *Backup and restore* and *Diagnostic and repair*. Refer to Jenkins configuration for more details.

MODIFIED: In addition of the categories reorganization, the jobs have been renamed to make it easier to sort them: *Stop server* and *Start server* have been renamed into *Server - Start* and *Server - Stop* for instance. Refer to Jenkins configuration for more details.

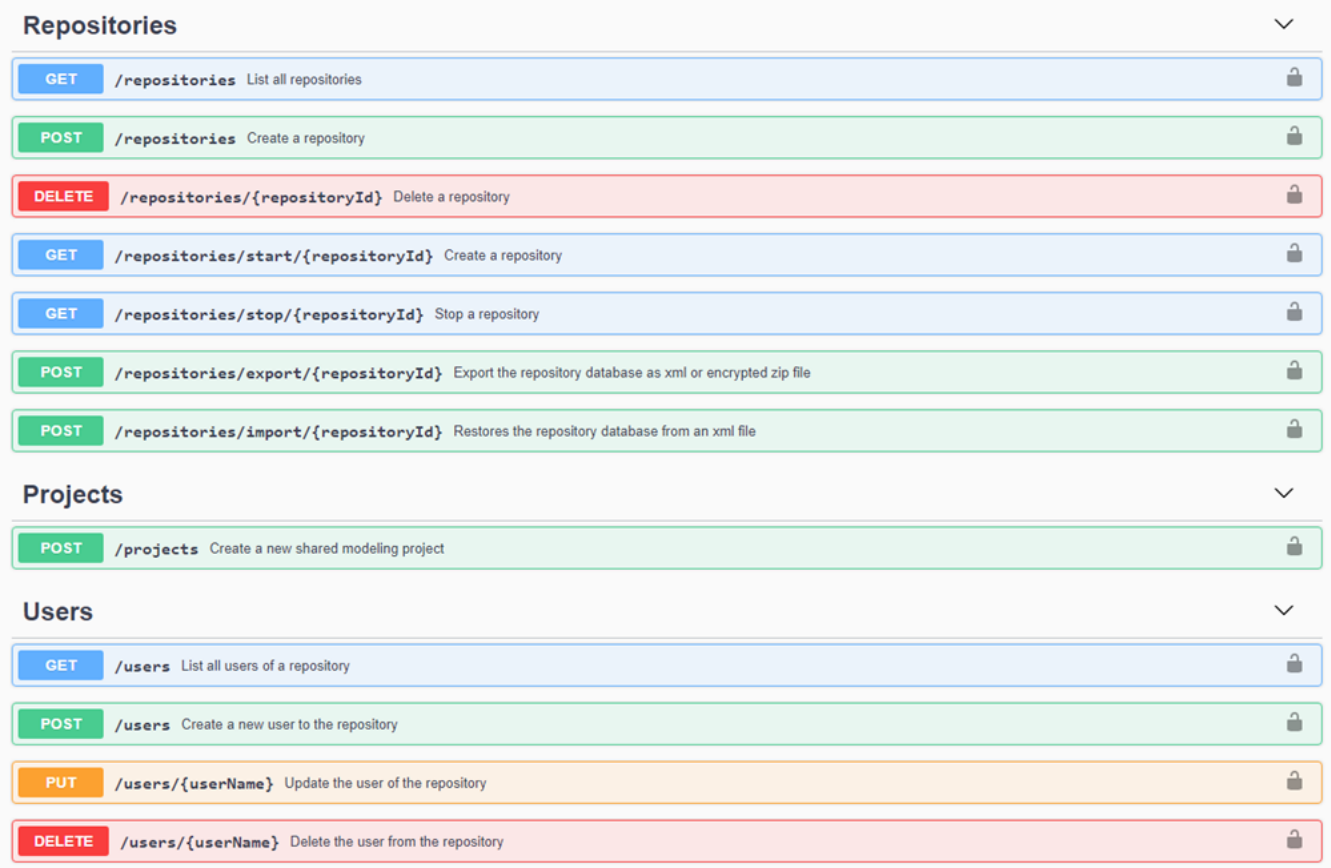
Changes in com.thalesgroup.mde.melody.collab.importer

ADDED: Importer application arguments management and validation have been improved regarding the different supported import strategies. For example, to configure an offline import (import from an xml backup of the repository), only the *outputfolder* and *XMLImportFilePath* arguments are required.

Server / Repository configuration

ADDED: The LDAP authentication support has been completed with the capability to use a manager account when the anonymous binding is disabled. Refer to Configure LDAP with a manager for more details.

ADDED: A new ***experimental*** administration feature is now available for the server, it brings users and repositories management capabilities through REST WebServices and exposes an OpenAPI description. Refer to documentation available in the folder server/experimental to discover how to install and enable it.



Compatibility with other add-ons

MODIFIED: The export of data to the server now tries to dispatch the feature extensions referencing a representation to the corresponding .srm resource. This behavior can be disabled with the system property `fr.obeo.dsl.viewpoint.collab.internal.export.move.feature.extension.srm=false`. Feature Extensions are M2 concepts from Eclipse Sirius which might be used by third parties add-ons to add model information to the .aird ressources.

MODIFIED: The import of data from the server now gets feature extensions stored in the .srm resource back in the resulting .aird.

Changes in Team for Capella 1.4.0 (from 1.3.1)

Please also refer to Sirius Release Notes, Capella Release Notes and Sirius Collaborative Mode Release Notes

ADDED: An application has been added to perform diagnostic and maintenance actions on a repository. It is declared in the new plugin `com.thalesgroup.mde.melody.collab.maintenance` and can be launched from the Scheduler's dedicated jobs. Refer to *Server Administration / Administration tools* section of the documentation for more details.

ADDED: The `fr.obeo.dsl.viewpoint.collab.server.warmup` plugin has been added on the server, it provides an `org.eclipse.emf.cdo.spi.server.IAppExtension` which reacts to repository start-up and loads all found resources which are direct children of the projects folder (.representation folder and .srm representation resources are excluded). This initializes the revision manager caches at repository start-up and speeds up the session opening of the first connection to each project. This behavior can be disabled with the system property

`-Dfr.obeo.dsl.viewpoint.collab.server.enabledWarmup=false`.

ADDED: A new import strategy has been enabled by default for the *Import projects* and *Import user profiles model* Scheduler's jobs. It allows to perform the import based on an XML extraction of the repository. In this mode there is no connection to the server and no interaction with other users, it also avoids to overload the server.

Partial support for internationalization

Team for Capella 1.4.0 introduces partial support for internationalization: all literal strings from the runtime part of the Team for Capella add-on are now externalized and can be localized by third parties by providing the appropriate "language packs" as OSGi fragments. Note that this does not concern the server components, the user profile component, the maintenance and importer applications, the administration components or the parts of the UI inherited from Eclipse/EMF/GEF/GMF/Sirius/CDO and other libraries and frameworks used by Team for Capella.

Some API changes were required to enable this. Most breaking changes concern the plug-in/activator classes from each bundle. They are:

ADDED: `com.thalesgroup.mde.melody.collab.license.registration.TeamForCapellaLicenseRegistrationPlugin`, a subclass of `org.eclipse.emf.common.EMFPlugin` has been added. The corresponding OSGi bundle activator is the internal class `TeamForCapellaLicenseRegistrationPlugin.Implementation`.

Additional non-breaking changes:

ADDED: The translation keys (and default values) have been added to all the concerned bundles, in their `plugin.properties` or `messages.properties` file depending on their initialization with `org.eclipse.sirius.ext.base.I18N` or inheritance to `org.eclipse.osgi.util.NLS`. These (translated) messages are available at runtime as static fields of `Messages` classes, added to all concerned bundles (always in the same package as their plug-in/activator class). The concerned bundles are:

- `com.thalesgroup.mde.melody.collab.ui`
- `com.thalesgroup.mde.melody.collab.license.registration`

MODIFIED: Existing `Messages` classes have been completed with additional translation keys (and default values). Mutliple `Messages` from the same plugins have been merged into a single class per plugin. The concerned bundles are:

- `com.thalesgroup.mde.cdo.emf.transaction`
- `com.thalesgroup.mde.melody.collab.ui.airdfragment`

MODIFIED: The translatable attributes from every `plugin.xml` have been have been extracted with default values in the corresponding `plugin.properties` files.

Changes in com.thalesgroup.mde.melody.collab.importer

ADDED: The Importer constant `com.thalesgroup.mde.melody.collab.importer.api.TeamImporterConstants.CDO_EXPORT` has been added to launch the `cdo export` command and use this file as base to execute the repository import. This

parameter should be used with XML_IMPORT_FILE_PATH to determine where the `cdo export` file should be saved.

ADDED: `-XMLImportFilePath` argument has been added to allow to use the importer from a file produced by a `cdo export` command from the CDO server. In that case, the importer will not connect to the current cdo server but will perform the import from a virtual cdo server based on the XML export. The expected argument is the file path to the cdo export result.

ADDED: `-cdoExport` argument has been added to make it possible to automatically perform the `cdo export` command and use the resulting XML file as described in `-XMLImportFilePath` above documentation. The default value is false. The `-XMLImportFilePath` argument is mandatory since the same file path is used to perform the XML import.

Changes in Team for Capella 1.3.1 (from 1.3.0)

Changes in com.thalesgroup.mde.melody.collab.importer

MODIFIED: The com.thalesgroup.mde.melody.collab.importer.ImporterApplication application has been reorganized into a generic part which has been moved into the `fr.obeo.dsl.viewpoint.collab.importer` plugin and a Team for Capella specific part. The code has been refactored and dispatched in the proper plugins. The previous version of the `com.thalesgroup.mde.melody.collab.importer` plugin did not declare any classes as explicit API, `com.thalesgroup.mde.melody.collab.importer.api.TeamImporterConstants` and `com.thalesgroup.mde.melody.collab.importer.api.TeamImporterCDOExporter` have been promoted to API classes.

ADDED: New parameters have been added to the Importer application. They are declared in `fr.obeo.dsl.viewpoint.collab.importer.api.ImporterConstants` and are inherited by `com.thalesgroup.mde.melody.collab.importer.api.TeamImporterConstants`:

Arguments	Description
-exportCommitHistory	Whether the Commit History metadata should be exported (default: true). If the value is false, all other options about the commit history will be ignored.
-includeCommitHistoryChanges	imports the commit history detailed changes for each commit (default: false). This option is applied for all kinds of export of the commit history(xmi, text or json files).
-importCommitHistoryAsJson	import commit history in a json file format. The file has the same path as the commit history model file, but with **json** as extension.
-overrideExistingProject	if the output folder already contains a project with the same name this argument allows to remove this existing project.

Arguments	Description
-logFolder	defines the folder where to save logs (default : -outputFolder). Note that this folder needs to exist.
-archiveProject	defines if the project should be zipped (default : true). Each project will be zipped in a separate archived suffixed with the date.
-outputFolder	defines the folder where to import projects (default : workspace). Note that this folder needs to exist.

DEPRECATED: The -archiveFolder argument has been *(deprecated)*. It defines the folder where to zip projects (default: workspace). The use of -outputFolder must now be preferred (and -archiveProject=true but true is its default value).

Changes in the Team4Capella Scheduler

- The Jenkins version has been updated to from LTS version 2.46.2 (LTS) to version 2.150.2 (LTS). This brings several important security fixes and also notable changes as documented in the (LTS release notes).
- The "Start Server" and "Start Licence Server" are now automatically triggered with a sixty seconds delay after Jenkins starts.
- Temporary files created and used by the scheduler are now placed in a temp subfolder instead of the temp folder of Windows.
- A success result has been added to commands executed on the server by the command.bat application, some of its commands were properly executed but without a "success" result they kept running until a 2 minutes timeout stopped it.

Repository Information Properties Page

The properties page (contextual action) on aird files of shared modeling project has a tab named *Repository Information*. This presents the connected repository information (location, port and name) as well as a list of connected users on the same repository.

Changes in Team for Capella 1.3.0 (from 1.2.1)

Please also refer to Sirius Release Notes, Capella Release Notes and Sirius Collaborative Mode Release Notes

Representation lazy loading

A new mode allowing lazy loading of representations is activated for shared modeling projects. It translates into much faster project opening because none of the representation data are loaded. The data of a representation are loaded only when the application requires it. Examples: open representation, copy representation, export representation as image etc...

DEPRECATED: Passing from one mode to the other requires to clean the database. Indeed, the lazy

loading of representations is linked to the fact that the representations are split in many resources in the database. Nevertheless, the application will work properly with a mix of split or non split representations.

Technically, the lazy loading of representations is activated with the preference `CDOSiriusPreferenceKeys.PREF_CREATE_SHARED_REP_IN_SEPARATE_RESOURCE` set to true by Team for Capella. It can be disabled with the use of a system property: `-Dcom.thalesgroup.mde.cdo.emf.transaction.enableRepresentationLazyLoading=false`. The representation content is stored in a dedicated srm shared resource. Note that representations in local Capella projects are still stored in the aird resource.

xmiids resource usage has been removed

uid is a new attribute on Sirius elements that are serialized in aird (and srm) resources. It is used as technical id for any element from the Sirius model which are stored in the aird (or srm) resources except for GMF notation elements. The old xmiids shared resource is no more used. Its role was to ensure that the xmi:id of elements were kept after export/import on the Team for Capella server.

Changes in com.thalesgroup.mde.cdo.emf.transaction

REMOVED: `com.thalesgroup.mde.cdo.emf.transaction.AirdCDOResourceImpl` was used for aird resource. It has been deleted and replaced by `fr.obeo.dsl.viewpoint.collab.internal.remoteresource.CachedObjectCDOResourceImpl`

Changes in com.thalesgroup.mde.melody.team.xmisupport

REMOVED: The whole `com.thalesgroup.mde.melody.team.xmisupport` plugin has been removed as it is not useful anymore.

Diff/Merge in Team for Capella

The limitation that came out in Team for Capella 1.2.x is no more effective. While comparing a local project to a connected project or between two connected projects, no differences will be shown between representations if they are identical.

Please have a look at Capella Model Diff/Merge Documentation for more details.

Audit Mode

The *Audit mode* is now active by default in the Team for Capella server. This mode aims to keep tracks of all versions of each object in the server database. It is required for comparing different versions of the model for example.

Please have a look at Audit mode for more details.

User Profile

User profile resource permission now can use a regular expression with spaces. If you used the %20 encoding to bypass this problem, then you must replace it by a standard space to make it work with the new version.

Change Management

The Commit History View has been improved to display a commit list related to the selection and also displaying the impacted elements of one or several selected commits. See the Commit View section in the user documentation of Sirius Collaborative Mode for more details about those changes: Commit History View.

The commit description dialog box is displayed if there is a warning associated to the commit description. A warning occurs when:

- the Mylyn choice is checked in preferences and no uncompleted task is selected.
- the default choice is checked in preferences, the CDO History is used (because no uncompleted task is selected) and the previous commit message was of type Mylyn. The user is then asked to either change the message or reactivate the Mylyn task.

Please have a look at Change Management for more details.

Changes in Team for Capella 1.2.1 (from 1.2.0)

Uid can be used instead of xmi:id to identify a representation

Uid can be used as technical id for representations in case when the XmiId synchronization is disabled.

Please have a look at Capella release note for more details about the usage of uid and the migration of models from previous versions to update uids.

Diff/Merge in Team for Capella in case of deactivating (by default) the XMIID synchronization

Because of the abandonment of using XmiID as the identification for representations and their elements while performing a Diff/Merge operation between 2 Capella projects, the graphical internal elements between two representations are technically not possible to be matched. It causes an impact while comparing and merging 2 projects in Team environment:

- While comparing a local project to a connected project or between 2 connected projects, it will always show that there are differences between representations although they are identical. The reason is that the abandonment of using XmiID made identifying and matching the internal elements of representations impossible.
- While merging a local project to a connected project, or vice versa, the content between 2 projects will be then identical but while comparing them again, it will always show that there are differences between representations as mentioned above.

Please have a look at Capella Model Diff/Merge Documentation for more details.

This XmiidsResource creation during export and its synchronization mechanism are now disabled by default. The system property `-Dcom.thalesgroup.mde.cdo.emf.transaction.disableXmiidsSynchronization=false` allows to re-enable it if needed.

Please have a look at VM Arguments > Disable XmiId synchronization for more details.

Durable locking is now disabled by default

The durable locking mechanism is now disabled by default.

Please have a look at Durable locks management view for more details.

Changes in Team for Capella 1.2.0 (from 1.1.x)

Changes in `com.thalesgroup.mde.cdo.emf.transaction`

ADDED: The constructor **com.thalesgroup.mde.cdo.emf.transaction.MelodyCDOImporter.MelodyCDOImporter(CDOTransaction)** has been added to provide a **CDOTransaction** (like the one created on test connection) for the execution of the import/export instead of creating a new one.

ADDED: The method **com.thalesgroup.mde.cdo.emf.transaction.MelodyCDOImporter.processChecksBeforeExecution(Set<EObject>, boolean)** to have some validations before authorizing an export with resource override. Here the validation checks if the exported project is not a Library project (as the one about to be overriden could be used in a different project).

Viewpoint native/legacy CDO mode

Please have a look at Release note for Sirius Collaborative Mode for more details.

CDO 4.6

Team for Capella is now based on CDO 4.6 (previous versions used CDO 4.4).

2.2. Metamodel changes

Changes in Team for Capella 7.x (from 6.x)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 6.x (from 5.x)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 5.x (from 1.4.x)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 1.4.x (from 1.3.x)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 1.3.1 (from 1.3.0)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 1.3.0 (from 1.2.1)

Metamodel changes in Capella

Please have a look at the Capella release notes.

Changes in Team for Capella 1.2.0 (from 1.1.x)

CDO generation mode for feature delegation

The default strategy for CDO generation concerning Capella meta-model has been changed from reflective feature delegation to dynamic feature delegation.

Metamodel changes in Capella

Please have a look at the Capella release notes.

Chapter 3. User Guide

Contents

3.1. Overview

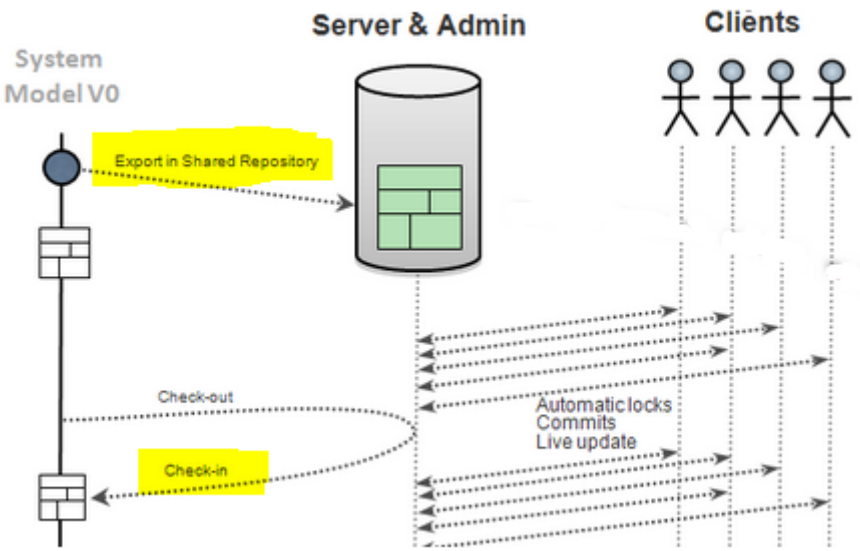
Team for Capella provides to its users additional functionalities on Capella projects allowing to collaborate easily thanks to:

- Simultaneous collaboration
  - Any object being edited is automatically locked and indicated to other users by a specific decorator. Only this object and its closest dependents are locked, allowing other users to continue working on the same model. These fine-grained locks are automatically released as soon as the modifications are saved. This allows several users working simultaneously on the same model.
- Instant updating
  - As soon as a modification on a model element is saved, it is automatically and instantly propagated across all users' views. No need to manually refresh your model to retrieve modifications performed by other users: you are always working on up-to-date model elements.
- Explicit locking
  - When a user needs to work during a long period on the same set of model elements, he can explicitly lock these elements. The lock will only be released on-demand, as soon as the owner of the lock decides to allow other users to work on these elements
- Sharing a local project
  - Modeling projects which are installed on your environment can be exported to the remote repository in order to be shared with other team members.
- Retrieving a remote project
  - Projects installed on the shared server can be manually imported into your environment or automatically saved to a backup server.

- Change history
  - History of commits is available to see which changes occurred on the shared models. At any time, you can compare two versions to see the differences. You can also see all the model elements and diagrams impacted by several commits.

3.2. Export/Import to/from the Team for Capella Server

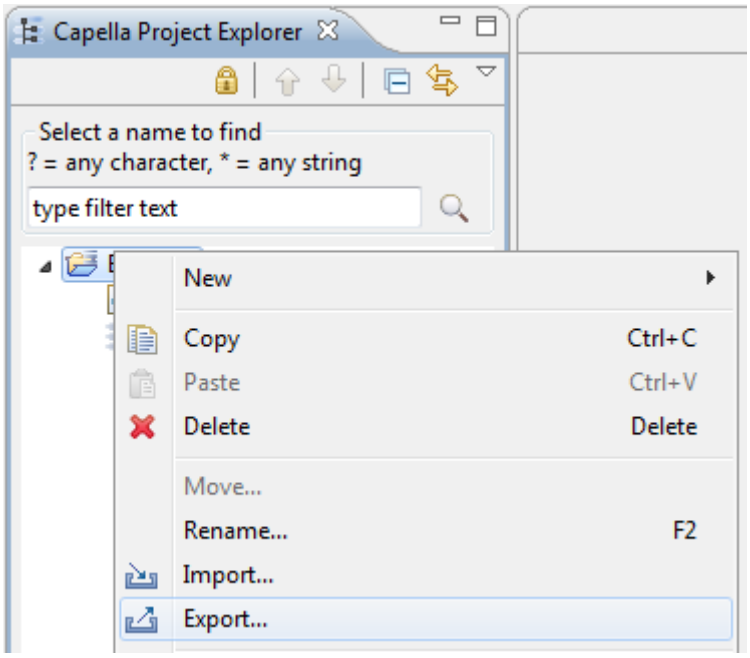
- **Export** is the action to put a model in file format on the Team for Capella Server
  - This model in file format may come from SCM tool (like Git). In that case, it should be a specific baseline.
- **Import** is the action to get a model in file format from the Team for Capella Server
  - It is necessary to push back a model in the SCM tool (a baseline can be put if a milestone has been passed).
- **Dump To local** is the action to save the already loaded resource of an opened connected project.



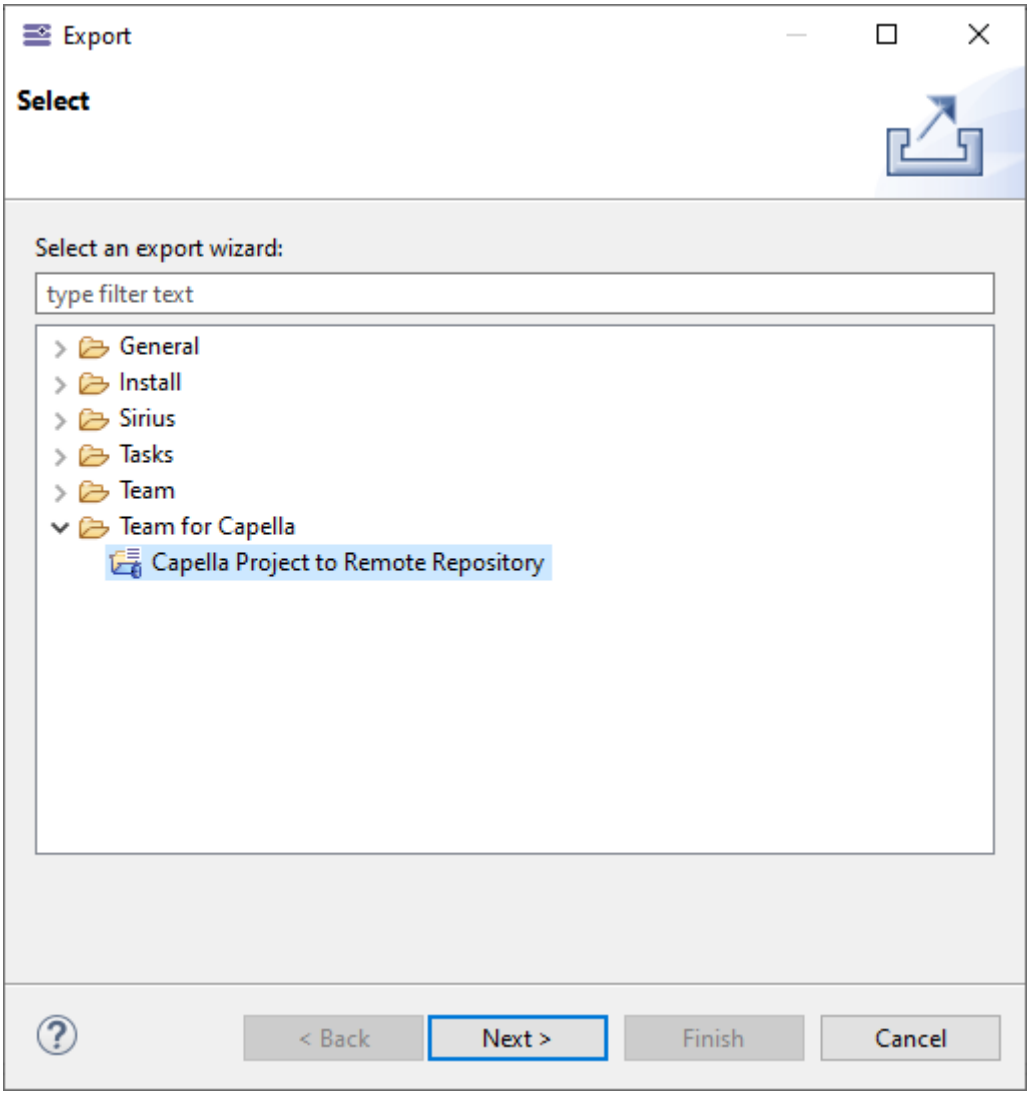
Export

Import a file-based model in a workspace. The model can be indifferently fragmented or not.

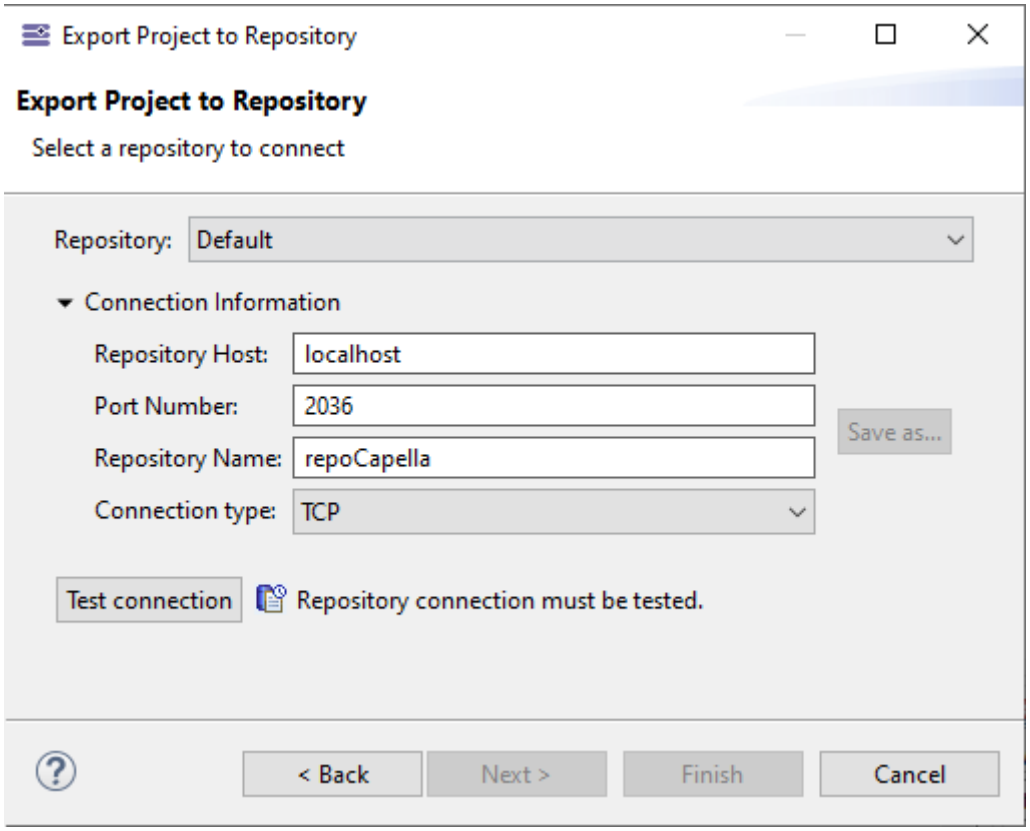
On the Capella Project containing the model, use the contextual menu to launch the Export wizard.



Choose "**Capella Project to Remote Repository**"

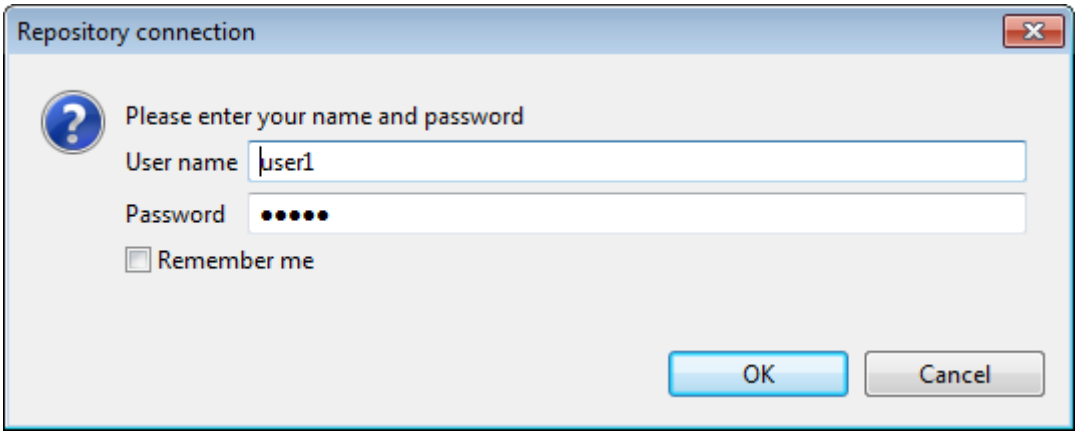


The "Export model to repository" wizard opens. The repository information is initialized with the default settings defined in the Preferences.

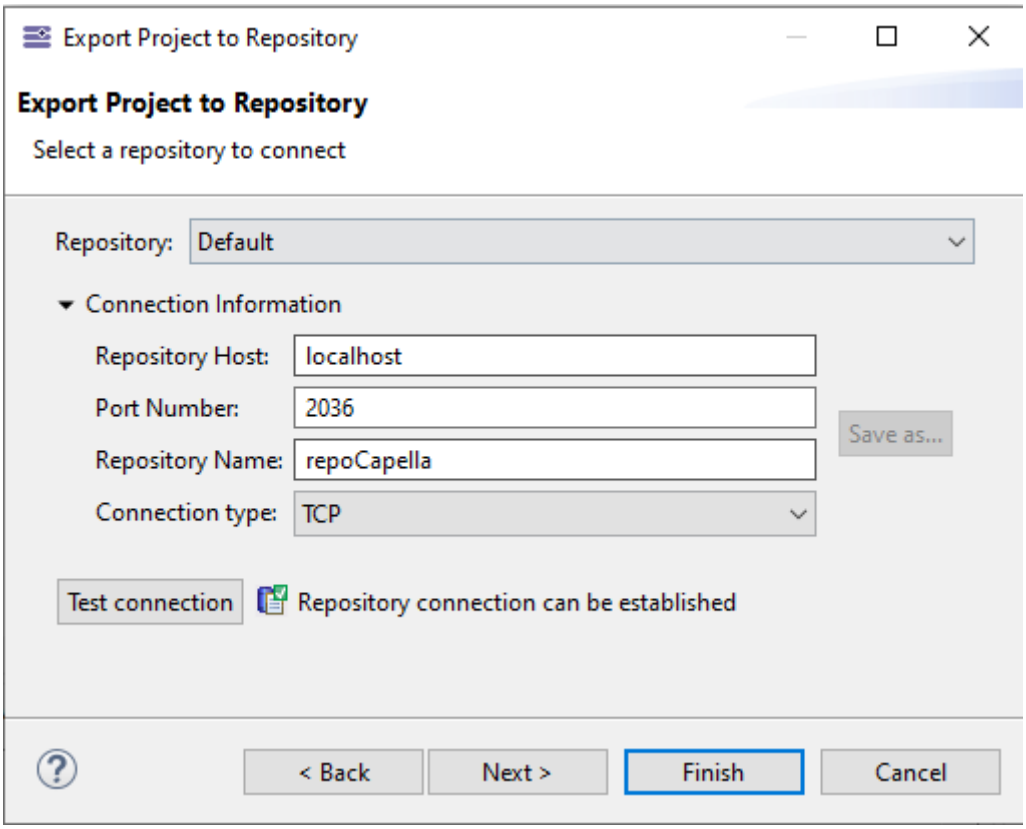


Before continuing, the server information has to be verified. To do so, click on "**Test connection**"

A login dialog pops up. Enter valid login and password (see Server Administration for more information about User management).

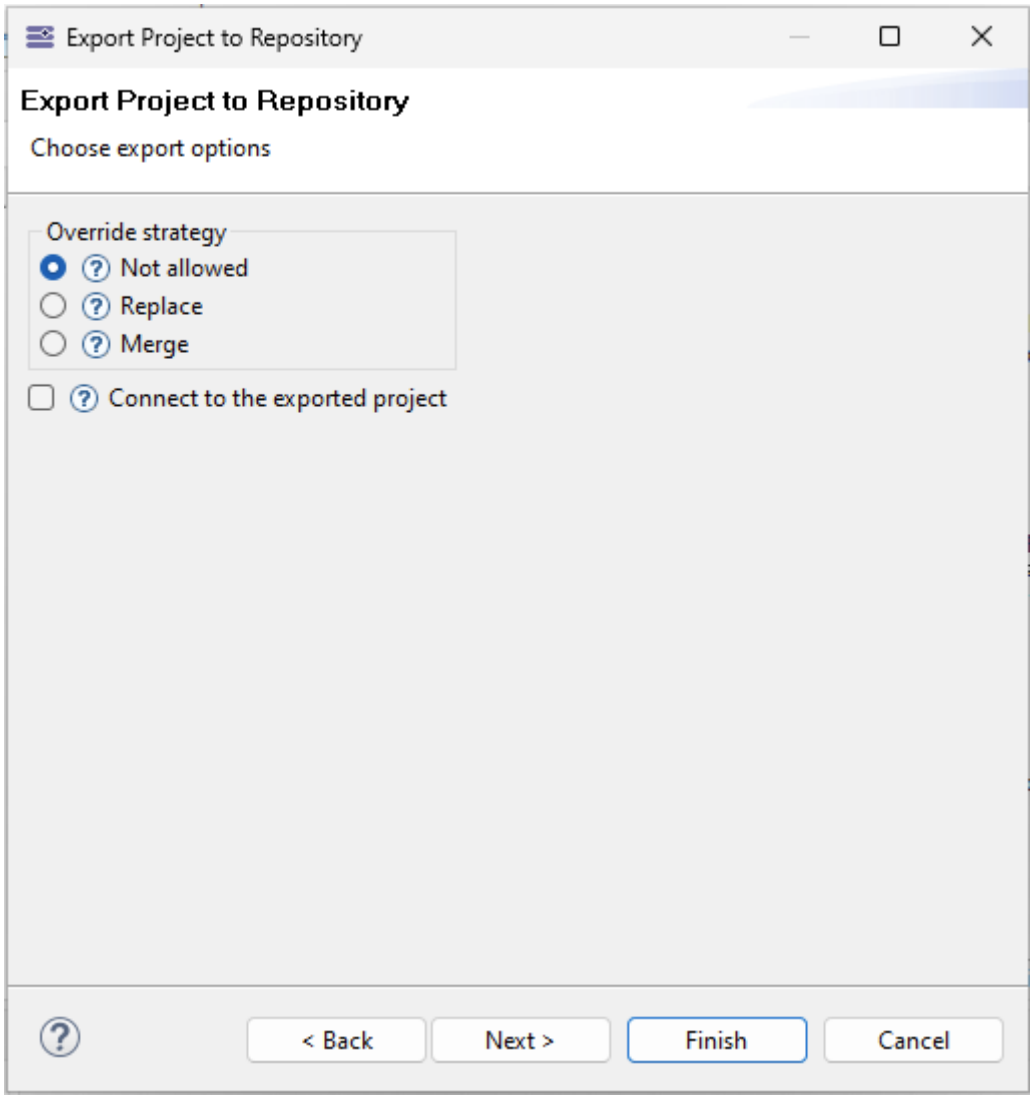


If the identification is successful, the "**Finish**" button becomes active.



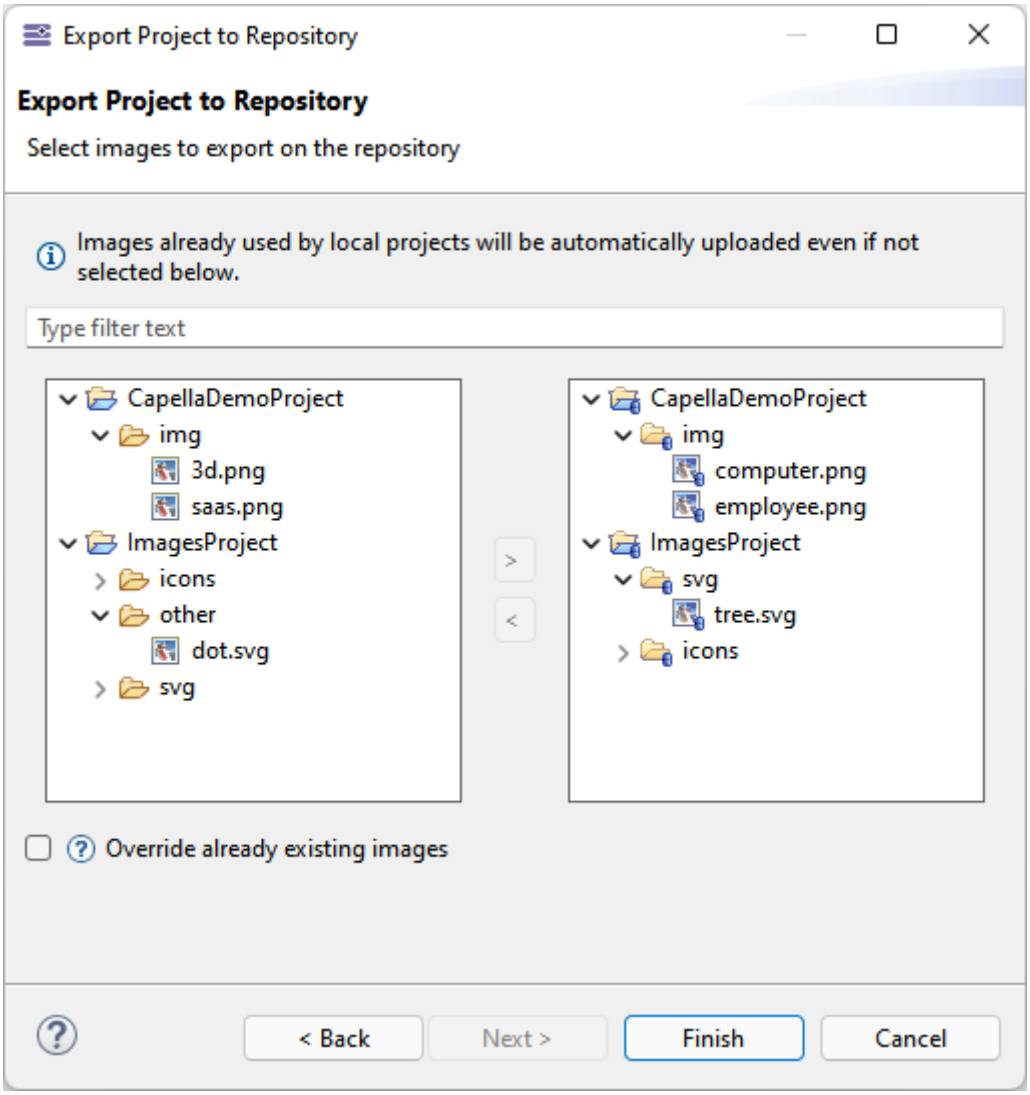
If you do not click on "**Finish**" but on "**Next**", the following options are available:

- Override strategy:
  - Not allowed: Default choice. Display an error if the repository already contains the same resources.
  - Replace: if the repository already contains the same resources, they will be overridden. This is done only if there is no other connected session and if it is not a library project. The remote project is deleted and replaced by a new export.
  - Merge: to use Diff/Merge to compare a local project and an existing remote project in order to commit only computed deltas. It keeps model elements identity and commit history.
  - Note that with the system property `-Dfr.obeo.dsl.viewpoint.collab.api.export.without.check=true` it is possible to override without checks.
- Connect to the exported project: this option causes to directly launch the connection wizard after the export wizard. This option is unchecked by default.



If you click "**Next**" again, you will be able to choose the images you want to export to the repository in this new wizard page.

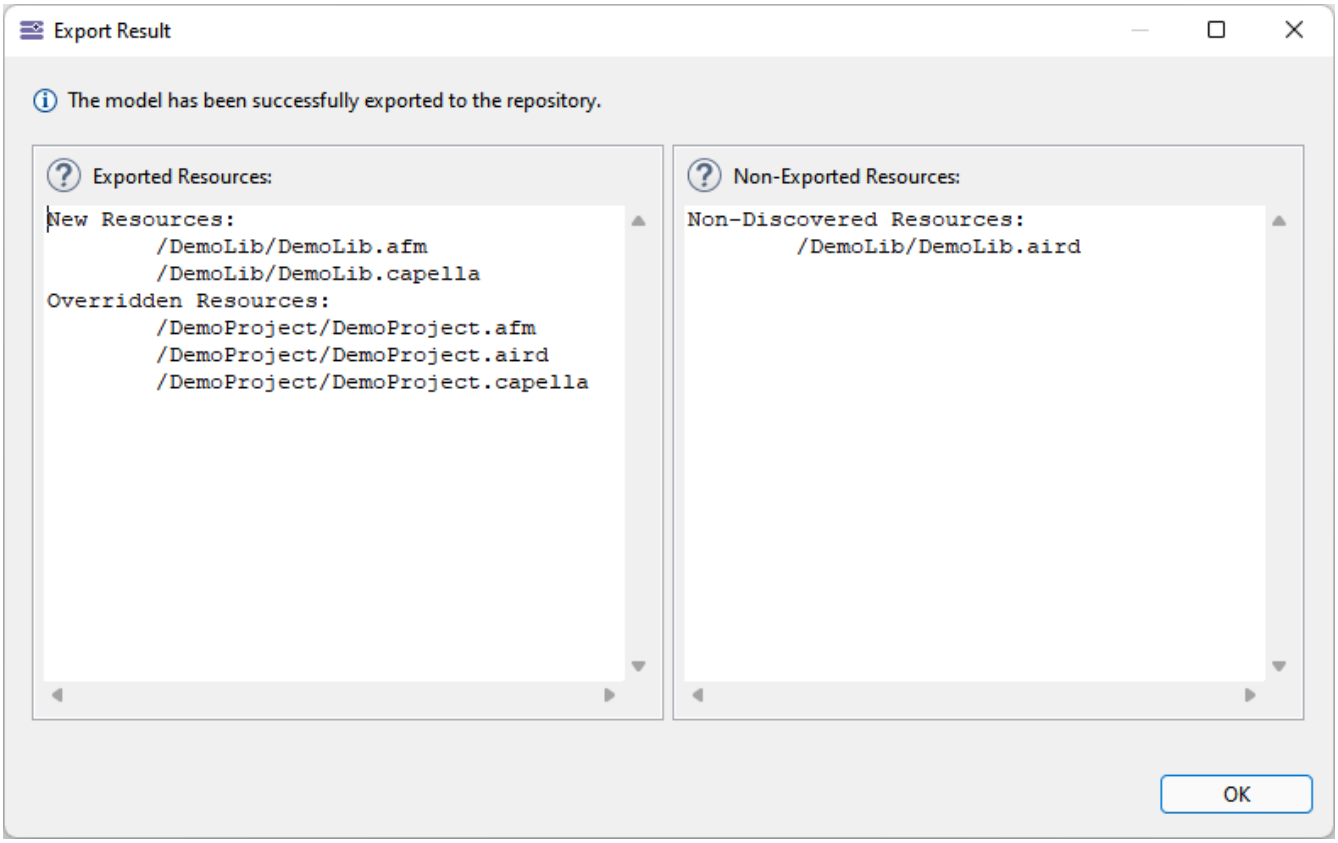
Refer to Export images to the server when exporting the project for more details.



Then, after having clicked Finish, a progress bar is displayed.

When the export is completed, a dialog shows the result of the process by listing the newly created or overridden resources, as well as the not found resources, already existing resources, or the non-discovered resources.

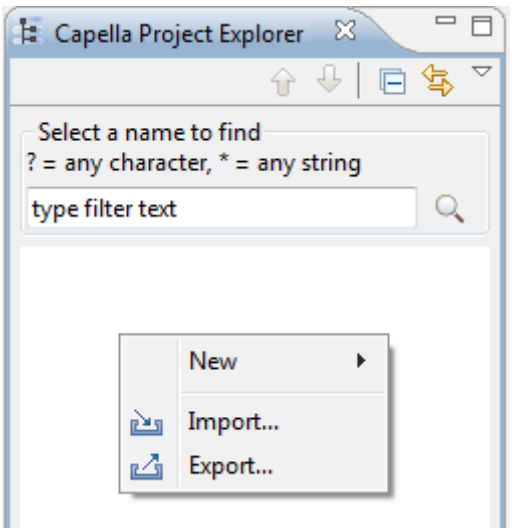
Note that the "discover" mode is not yet implemented, but this dialog allows informing the user about what has been done during the export.



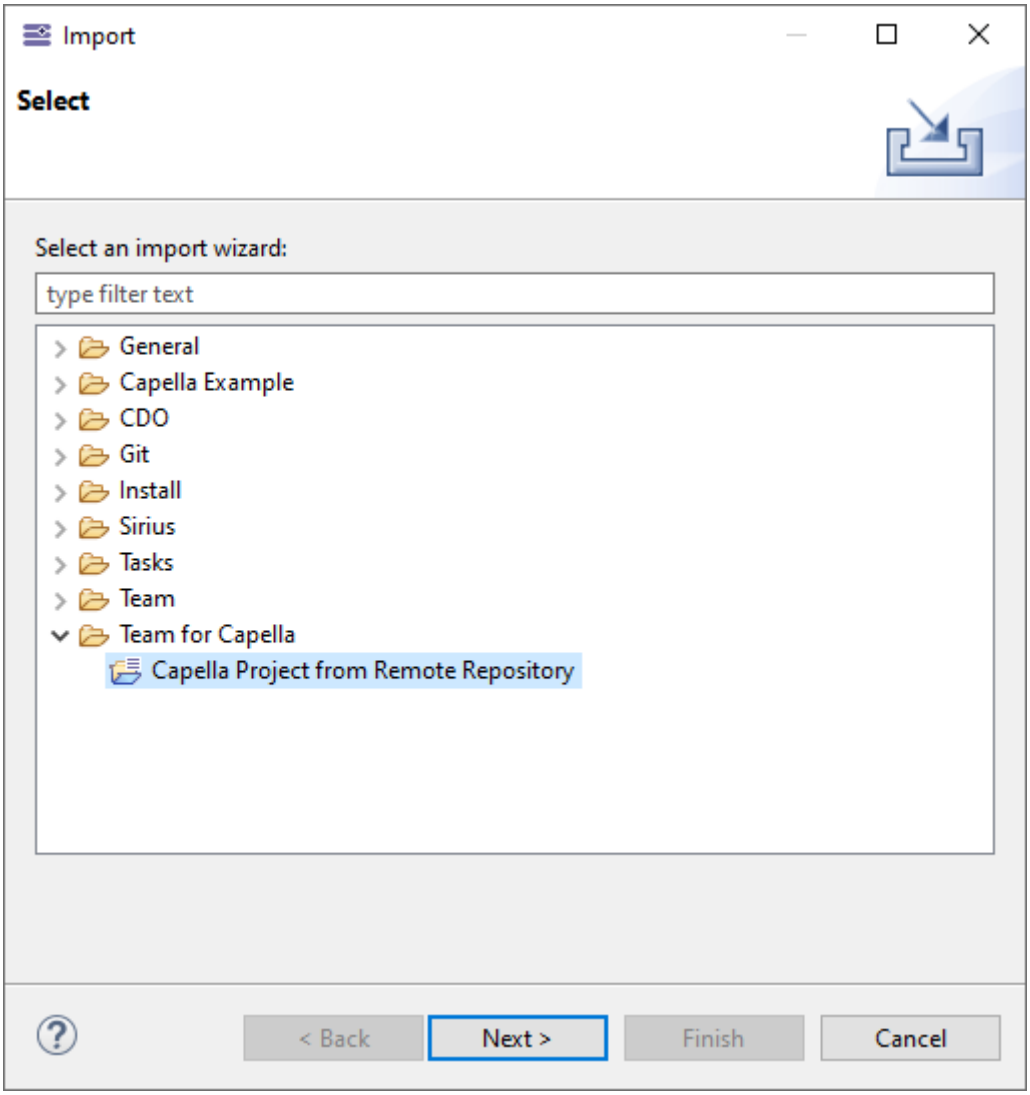
The CDO framework can only handle one version of a metamodel. If you shared a Capella project of one version (for instance Capella 6.1), you will not be able to export on the same repository a Capella project of a different version (for instance Capella 7.0). You will need to clear the repository or export on a new one. There is the same limitation with Capella Addons that extend the Capella Metamodel. Note that there are some corner cases where it works, for instance if the new metamodel version only adds new metaclasses, but in general it should be assumed that a new version of a metamodel should be deployed on a new repository.

Import

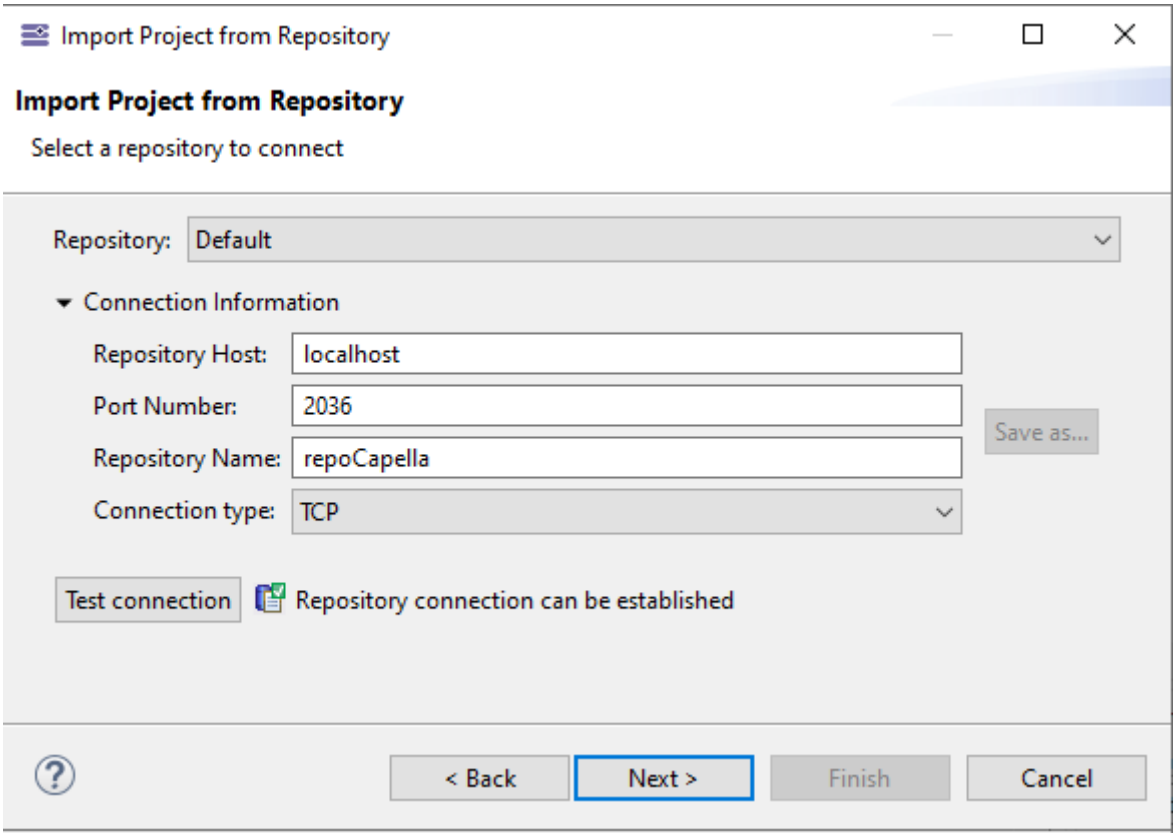
In the Capella Project Explorer, use the contextual menu to launch the Import wizard.



Choose "**Capella Project from Remote Repository**"



A wizard opens. The repository information is initialized with the settings defined in the Preferences. These information can be overridden. Before continuing, the server information has to be verified. To do so, click on "**Test connection**". Follow the login instructions as when login to Export the model. When the test is successful, the "**Next**" button becomes active.

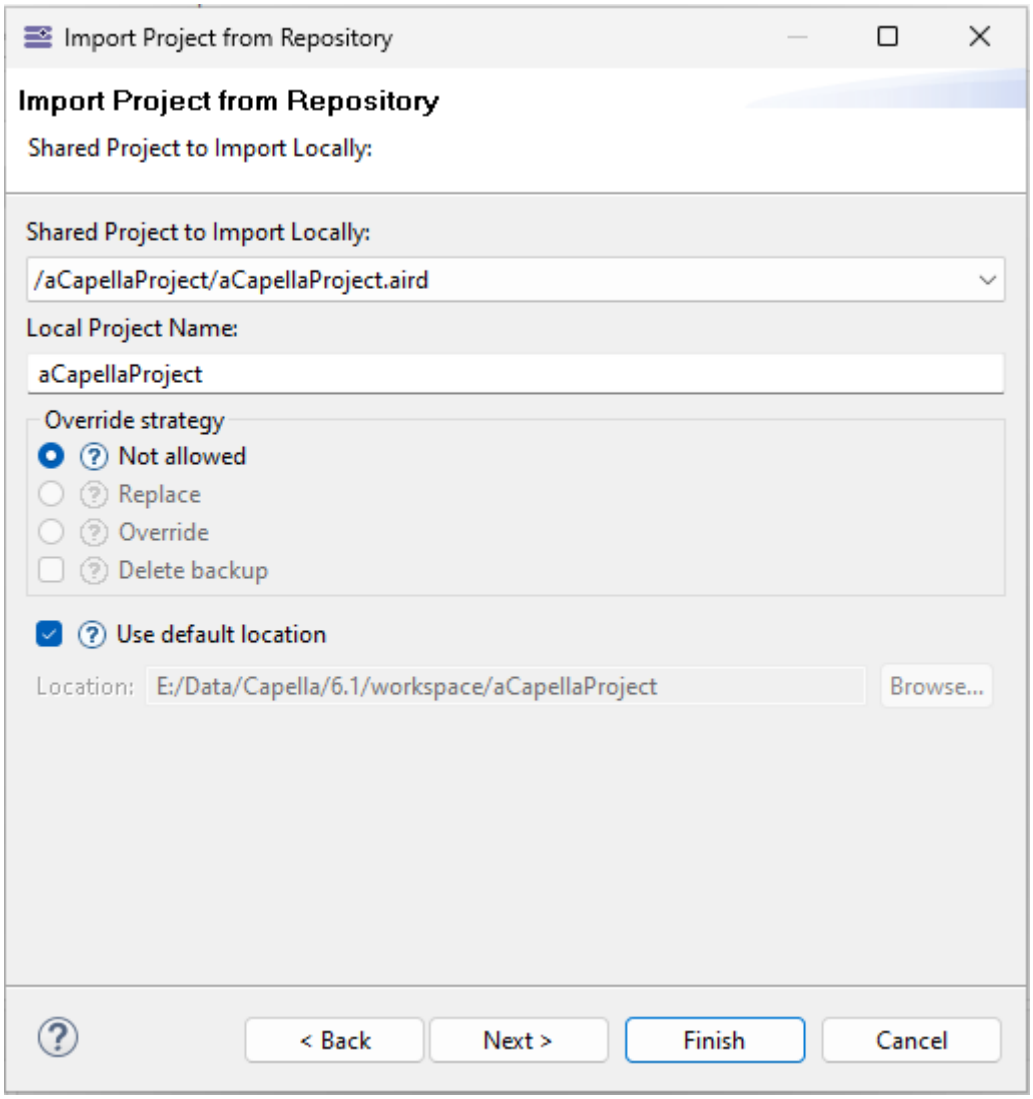


A second Wizard page proposes to choose the model to Import (a Shared Repository can hold several models).

Optionally change the name of the Capella Project going to be created.

The behavior of the wizard can be configured with the following options:

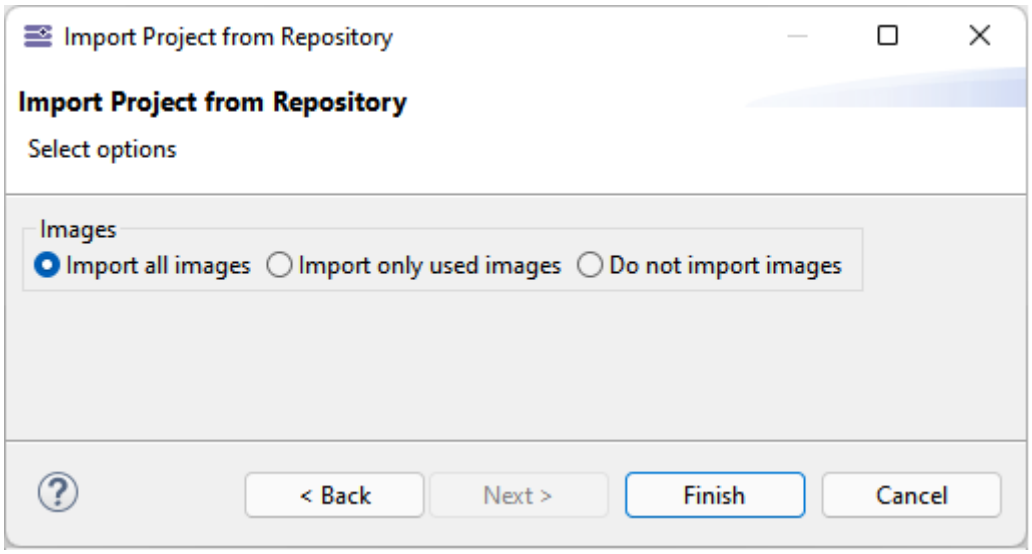
- Override strategy:
  - Not allowed: Default choice. Display an error if a resource to import already exists in the workspace or on the file system.
  - Replace: The local project is emptied, and its content is replaced by the remote project.
  - Override: Override local content with their remote equivalent, files which have no equivalent in the remote are untouched.
- Use default location: when unchecked, the location field allows selecting a folder on the file system which can be different from the current workspace. This option is checked by default.



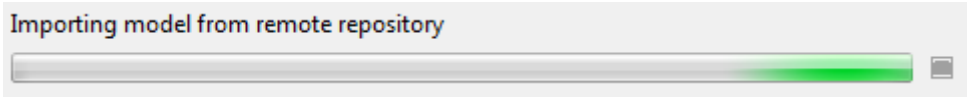
If you click on Next, you will be able to choose options about which images will be imported.

Refer to Import images from the server when importing the project for more details.

Images that already exist in the workspace will be overridden automatically.

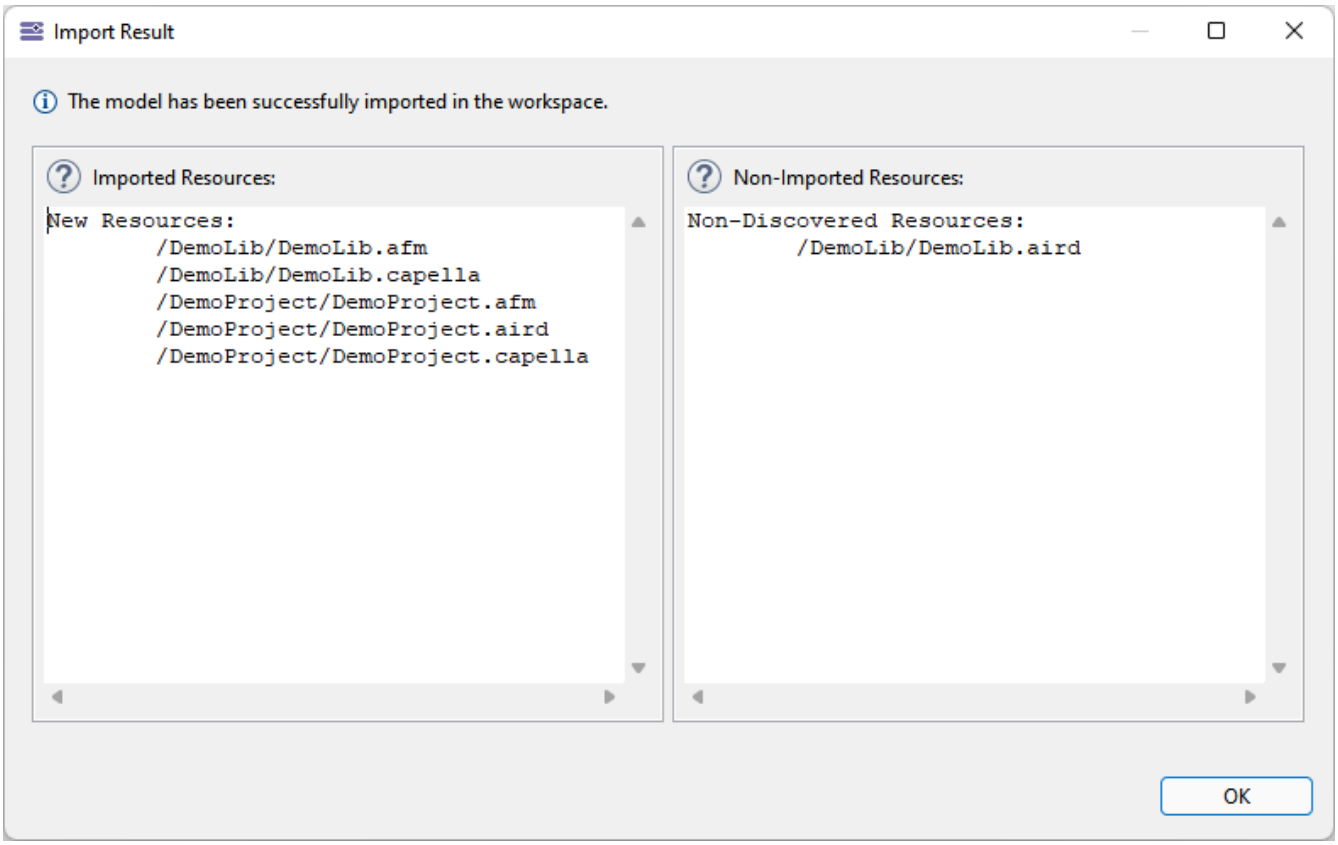


A progress bar appears.

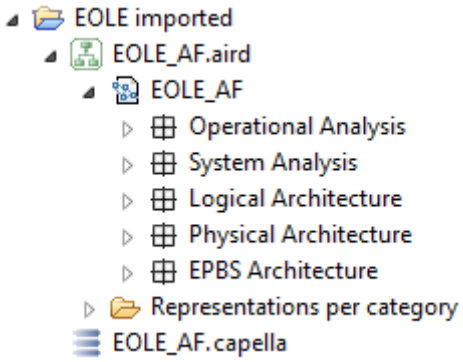


When the import is completed, a dialog shows the result of the process by listing the newly created or overridden resources, as well as the not found resources, already existing resources, or the non-discovered resources.

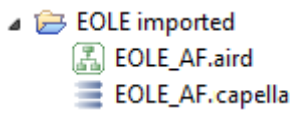
Note that the "discover" mode is not yet implemented, but this dialog allows informing the user about what has been done during the import.



Once the import is finished, the imported model is automatically opened.



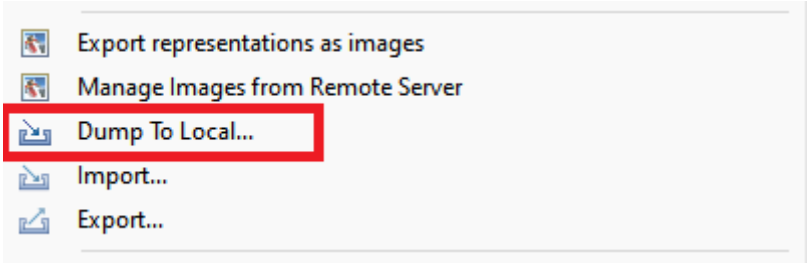
The model files can then be pushed back to Git if necessary.



Dump to local

This command will dump the connected project into a new local Capella project. The local project will contain only the already loaded representations.

It is available in the contextual menu on aird file of an opened connected project.



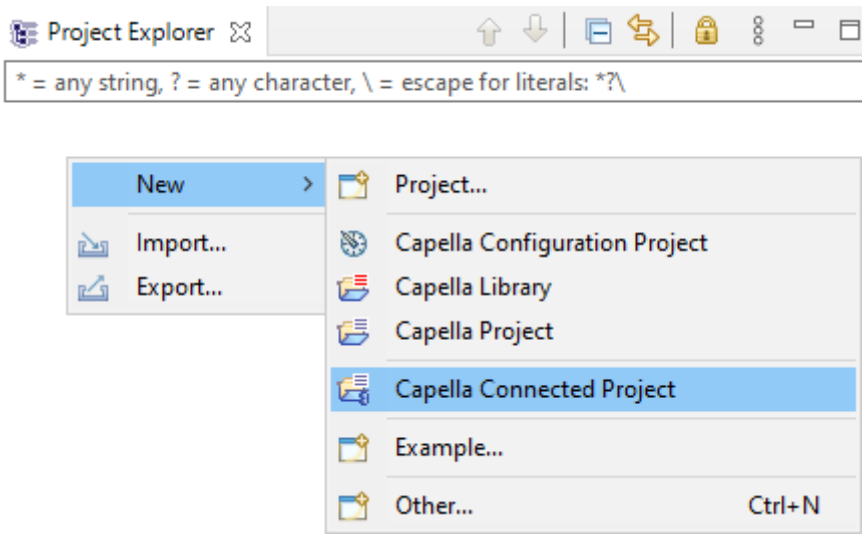
This command is useful if you encounter a Save fail issue. You can then use the tool to have a new Capella project, compare it with the project on server and make some merge.

3.3. Capella Connected Project

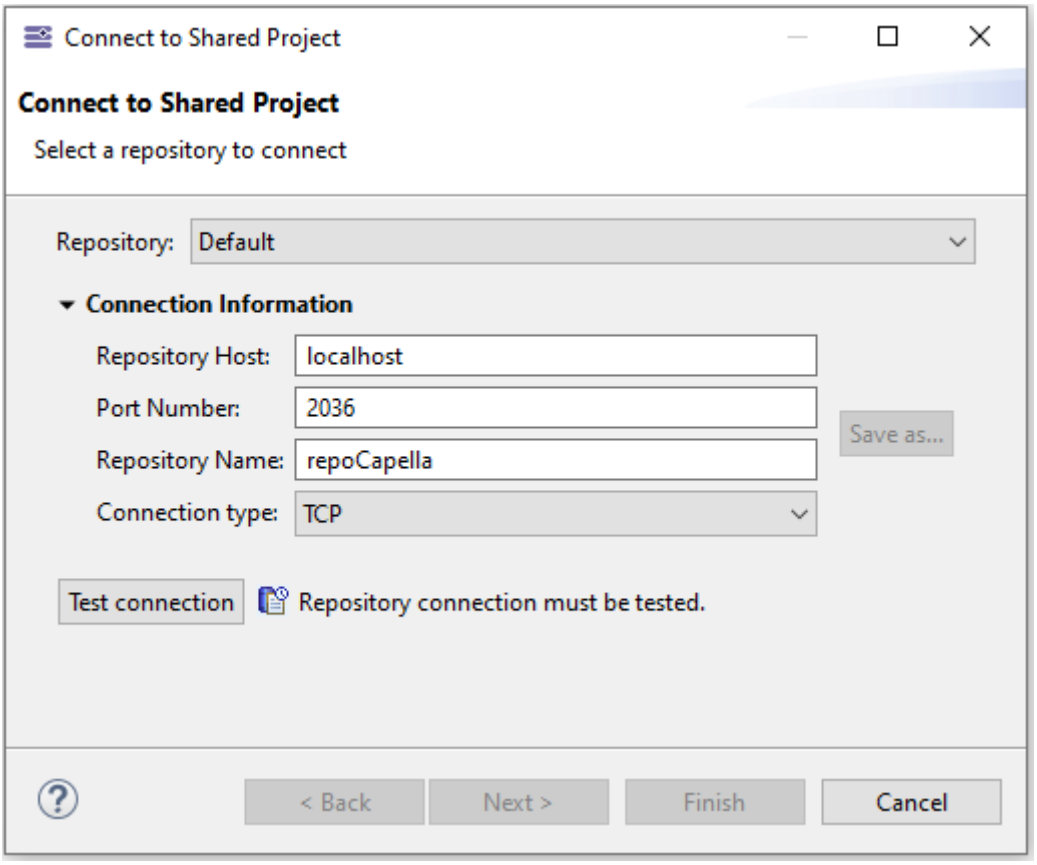
First Connection

Connecting to a remote model is similar to opening a file-based model. The result of a connection is an opened model ready to be modified.

Using the contextual menu on the Capella Project Explorer, click on New / **Capella Connected Project**

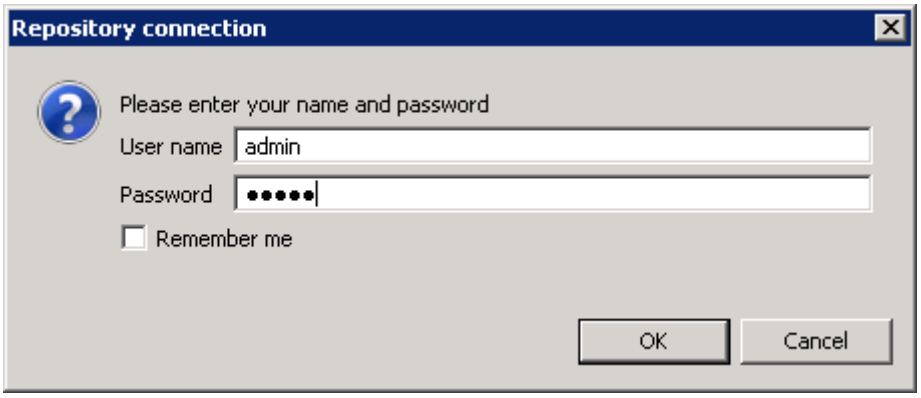


A dialog pops up, asking to specify the information of the remote repository holding the model. By default, these fields are initialized with the values set in the Preferences.



At this stage, the server information has to be verified. To do so, click on "**Test connection**".

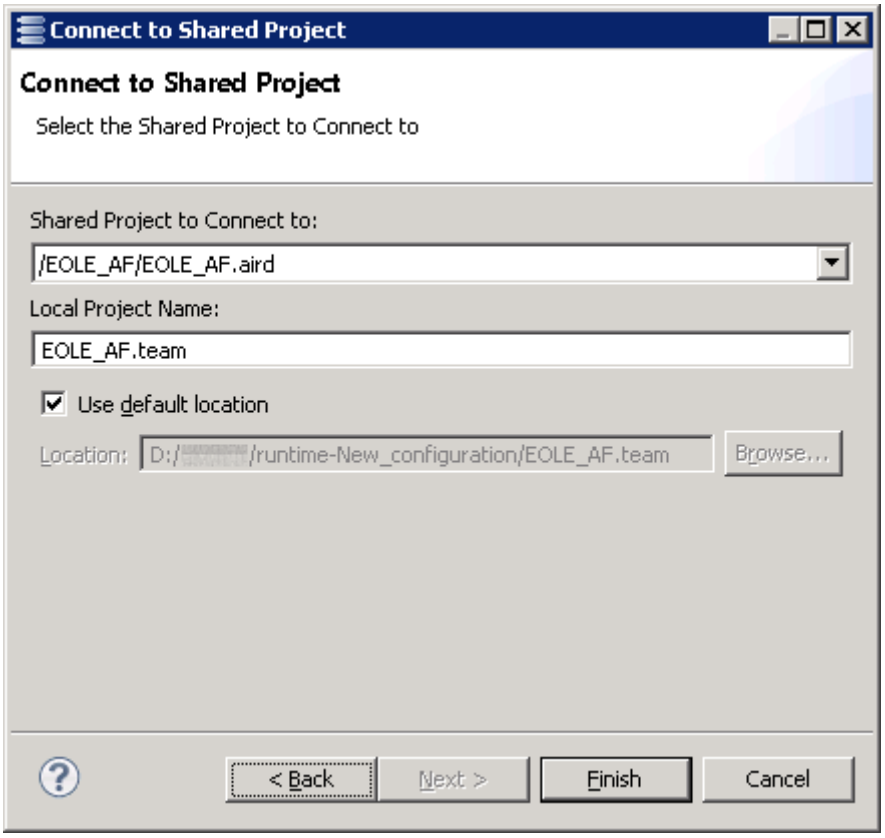
A login dialog pops up. Enter valid login and password (see Server Administration for more information about User management).



By checking "Remember me", you can store your username and password in the Eclipse's Secure Storage. If you do so, your username and password will not be asked for future connections.

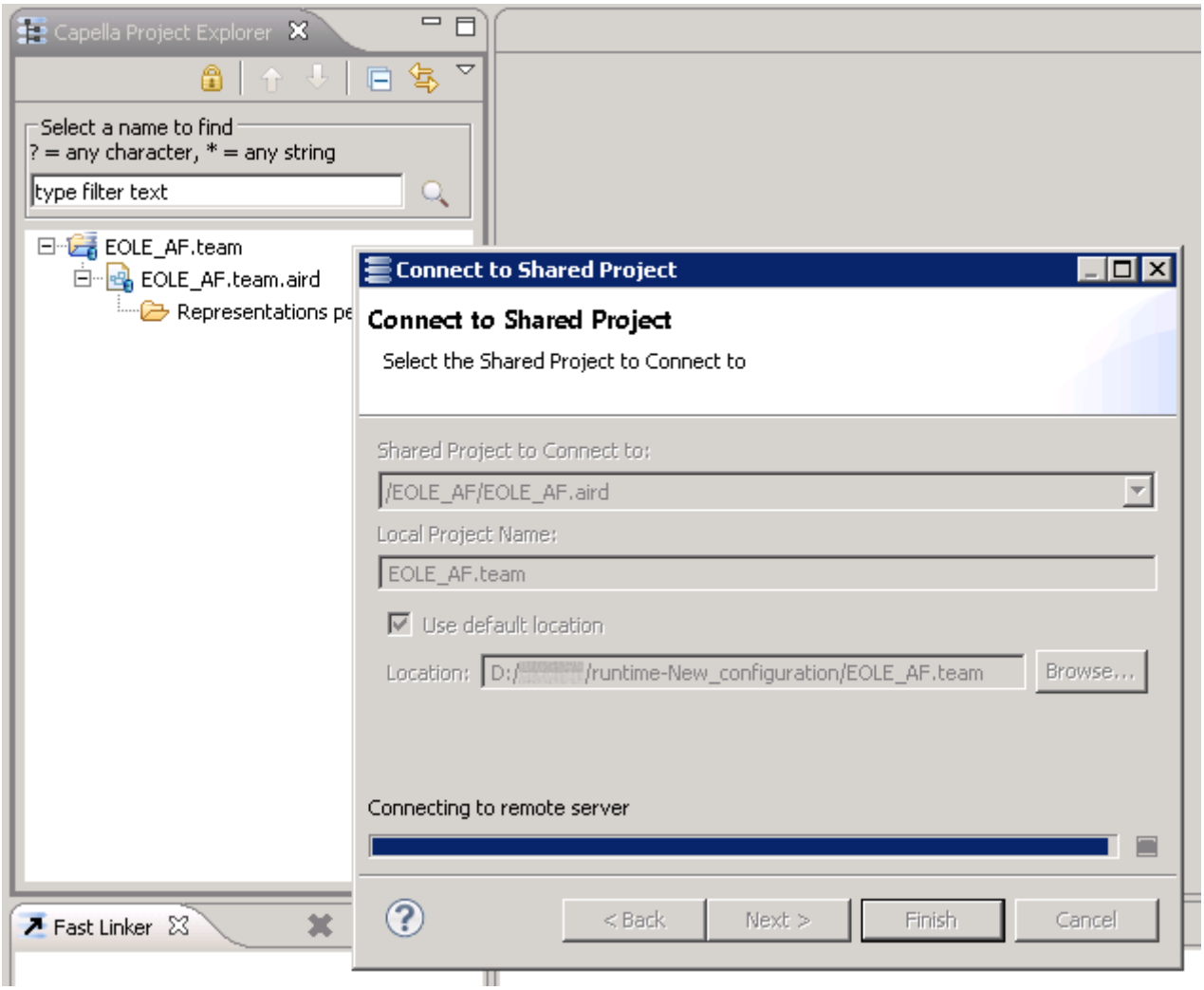
Once the connection is verified, click on "**Next**". Select one of the model hold in the repository.

The connection will create a new Capella project to hold the local proxy for the remote model. A suffix like ".team" is added by default at the end of the project name, to distinguish local and shared projects at the first glance.

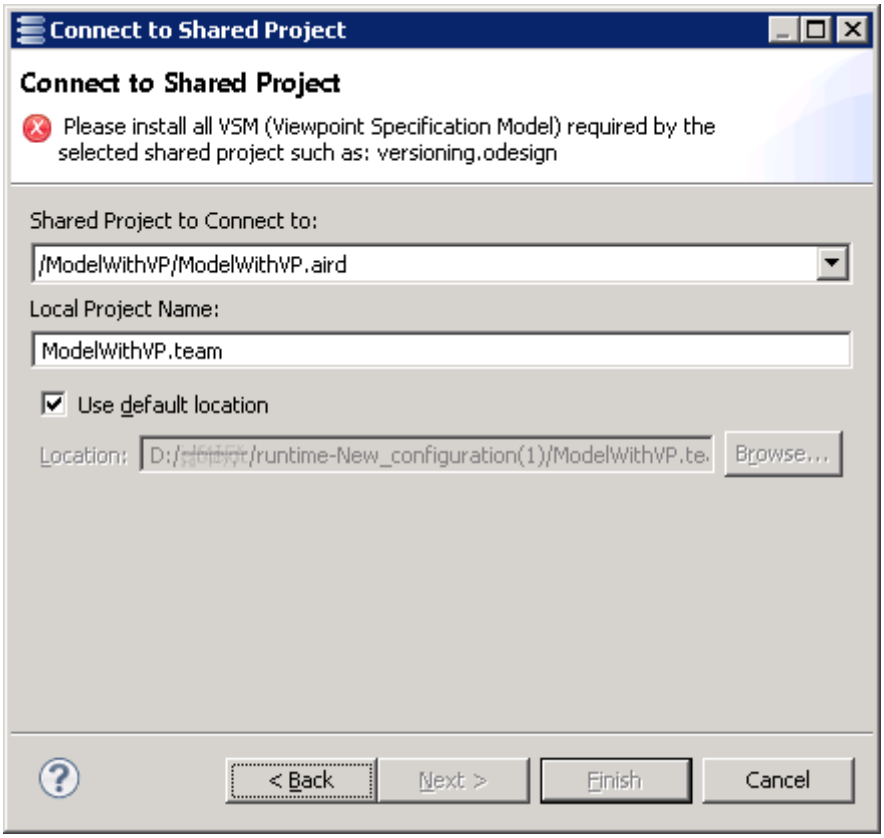


Click on "**Finish**". According to the size of the model, the duration of the connection may vary.

Warning: it is longer than opening a file-based version of the same model.

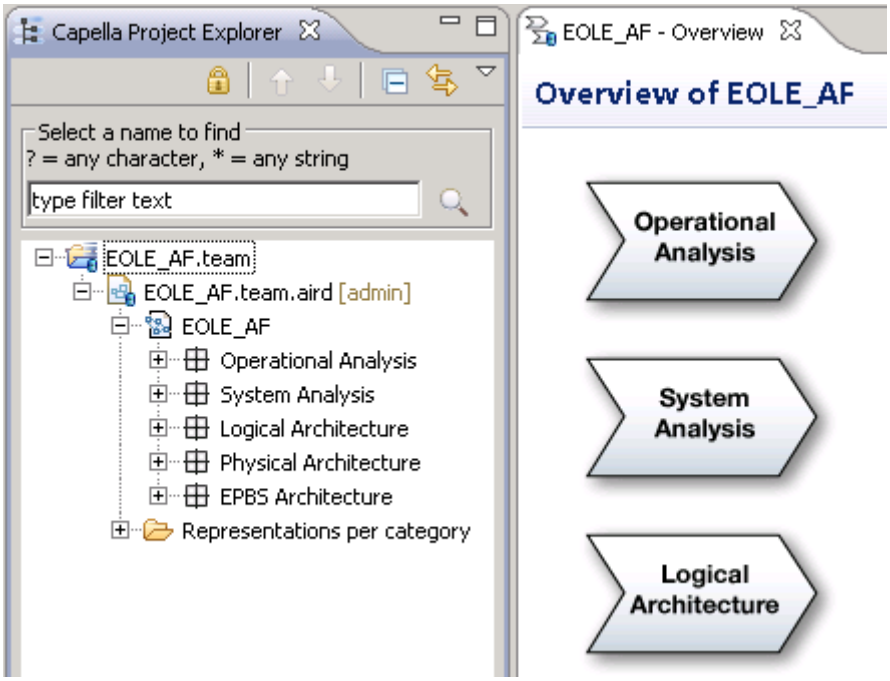


The connection can fail, for example, if a Viewpoint used by the remote model is missing on the client side. In this specific case, the following error will be displayed:



Known issue: if this error occurs, it is advised to restart Capella before trying to reconnect (even if you want to connect to another model for which there are no missing Viewpoints).

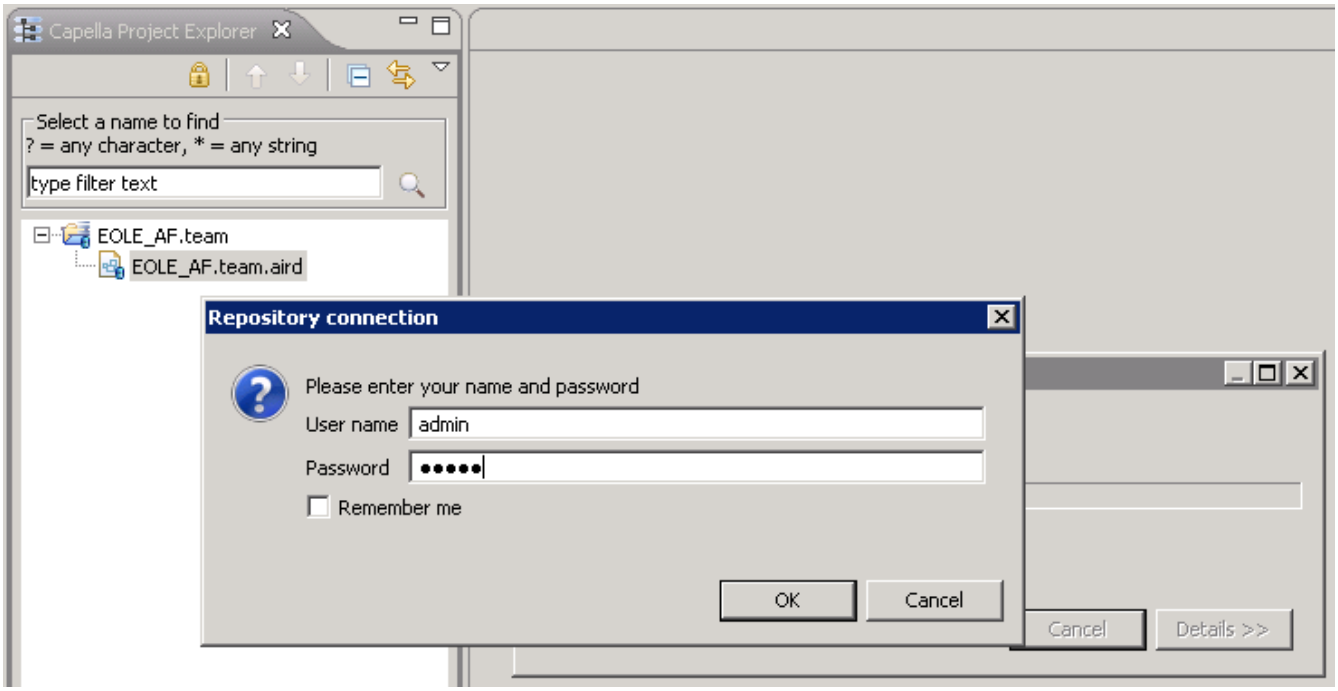
If the connection is successful, the model is opened in the Capella Project Explorer. Note there is no semantic file ".capella". The ".aird" file contains both information about the remote model and the local diagrams on this model.



At the end of a working session, the model can be closed exactly like a file-based model.

Connection Using an Existing Connection Project

When a connected project already exists, connecting again simply requires a double click on the ".aird" file. If necessary, the login dialog will be displayed.

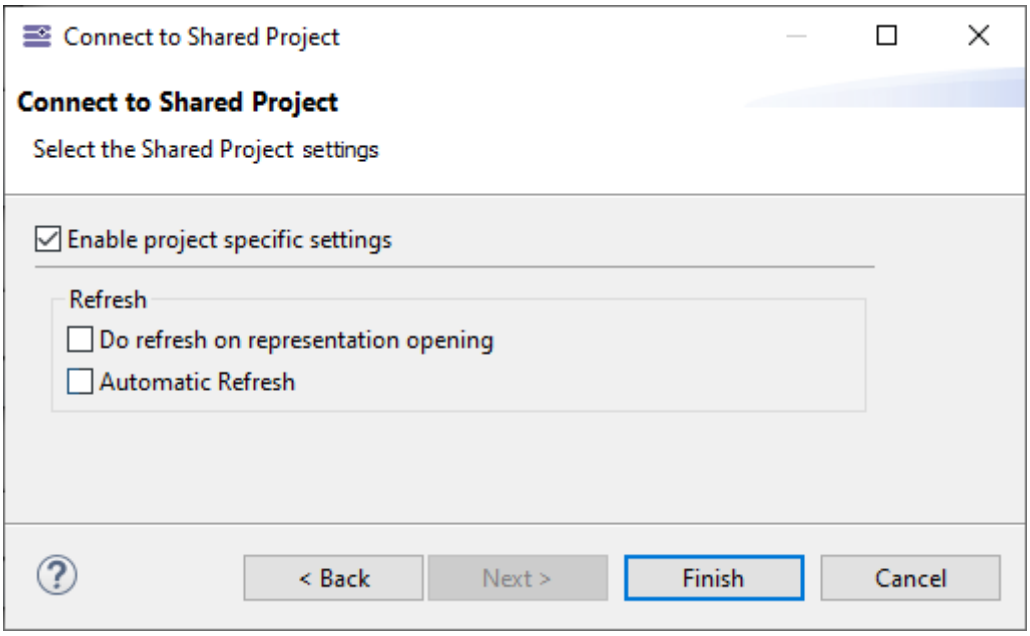


Overriding Sirius refresh preferences for a particular connected project

Both "Automatic Refresh" and "Do refresh at representation opening" can be specified for a given aird. Refer to Sirius documentation: Preference associated to the aird file

For any new local Capella project, the preferences are not overridden for the aird file and the preference values are those displayed in **Windows/Preferences/Sirius**

For a **connected project**, to define specific Refresh preferences, a page has been added in the **"Capella Connected Project"** wizard to allow users to override refresh preferences for the being created connected project local aird. By default, "Enable project specific settings" is checked and both "Automatic Refresh" and "Do refresh at representation opening" preferences are set to false.



It is nevertheless possible to change the default value using the preference **fr.obeo.dsl.viewpoint.collab/PREF_ENABLE_PROJECT_SPECIFIC_SETTINGS_DEFAULT_VALUE**. If set to false, then, by default, "Enable project specific settings" is unchecked.

Note: The preference values are not shared between two connected users. The preferences are associated with the local aird of the "Connected project" but not with the shared aird.

Tips and Tricks

Secure Storage (Remember me) and Roaming User Profiles

When "Remember me" is used, the login/password couple is stored in an encrypted file (located here: %USERPROFILE%\.eclipse\org.eclipse.equinox.security\secure_storage).

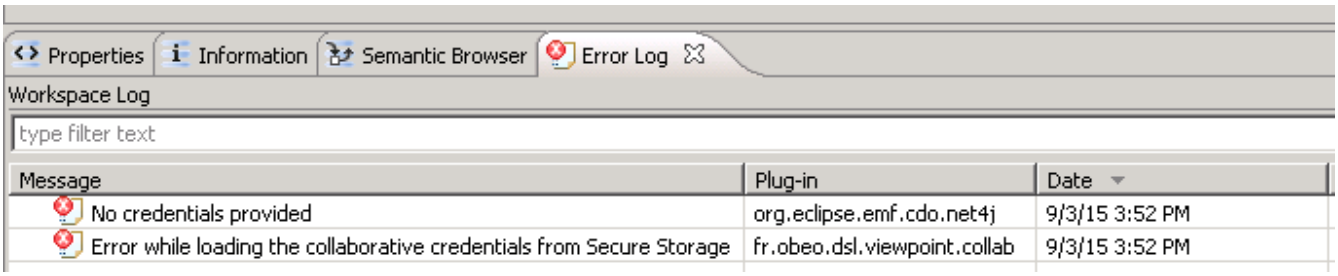
The key used to encrypt this file is generated and depends on the computer, the current Windows account and the Team for Capella architecture (32 bits or 64 bits).

So by default, this file can only be decrypted and used using the same computer/windows account/Team for Capella architecture (32 bits or 64 bits) than those used to create the file.

Because of this, it is not possible to use the Secure Storage feature with roaming user profiles.

Example: if the file was created using "Computer1"/User Account/Team for Capella 32 bits, it won't be possible to reuse the Secure Storage with "Computer2" or with another user account or with Team for Capella 64 bits.

In the cases described above, the following error will appear in the "Error Log":



A workaround for this problem is to provide, by configuration, the key to use to encrypt the Secure Storage file. To do that:

1. **Create a text file and put a key in it (you are free to choose any key),**
2. **Add the following parameter in the capella.ini file (before -vmargs):** `-eclipse.password <path to your key file>`
3. **Then clients must clear their existing Secure Storage (if any) by using the procedure below and restart Team for Capella.**

How to Clear the Secure Storage

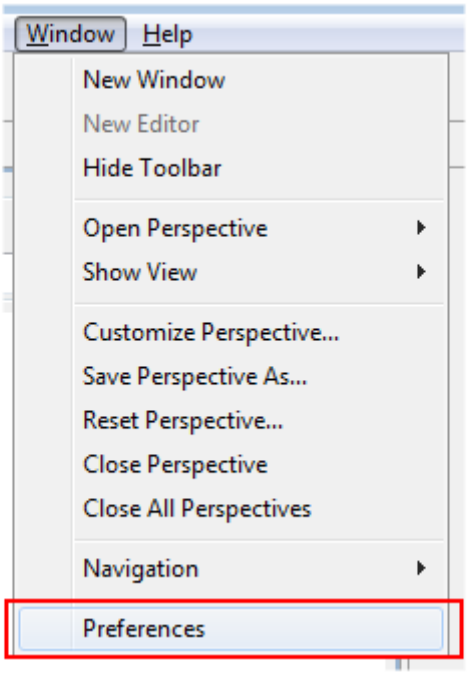
In the following cases, it could be useful to clear the Secure Storage:

- **A login/password couple is stored, and you do not want to use it anymore,**
- **An incorrect login/password has been stored in the Secure Storage and you are stuck with it**

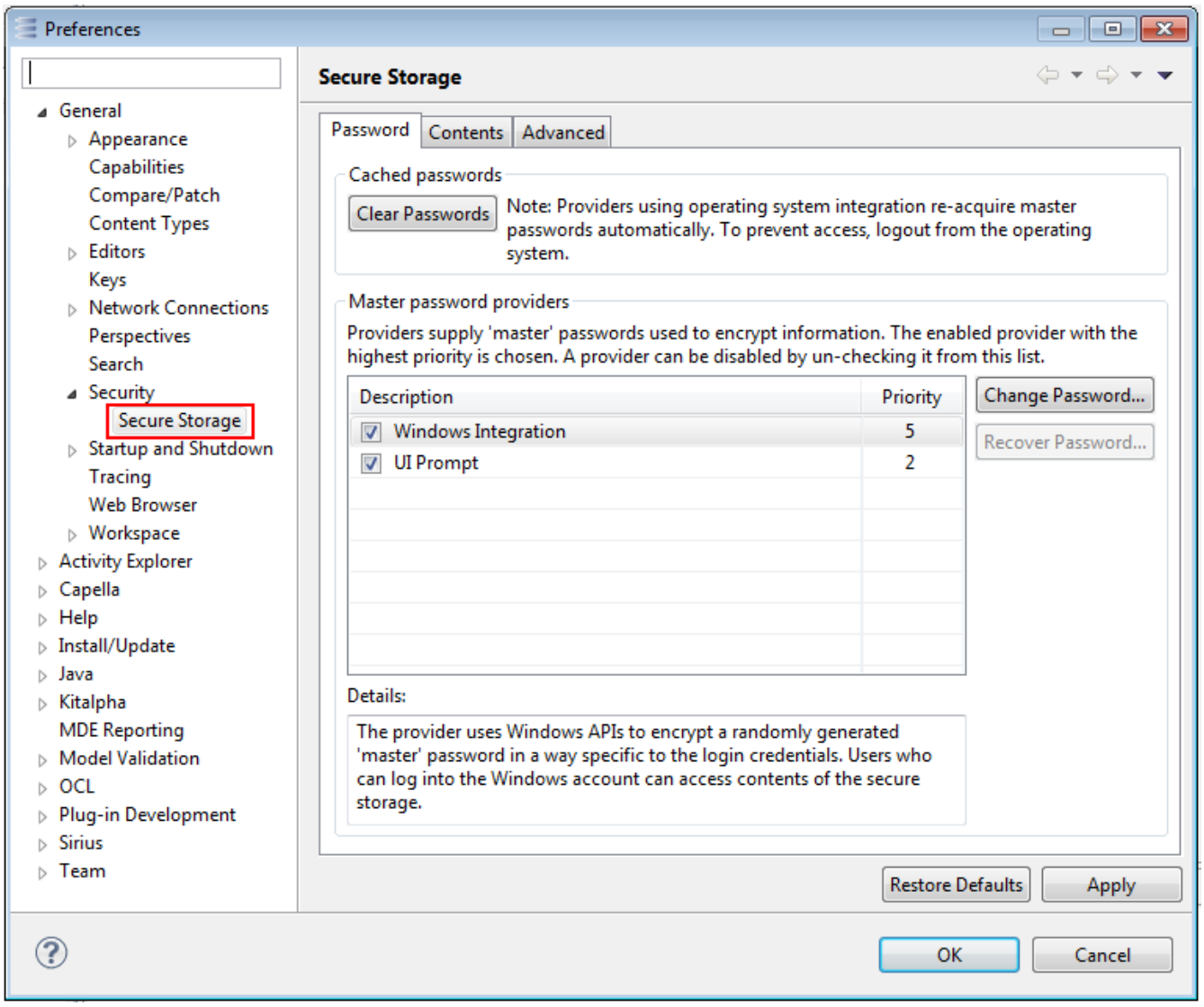
To clear the Secure Storage:

Note: It is not possible to just reset a stored username and/or password for a single repository. By performing these actions, the entire password store will be deleted, and you will then have to re-enter your username and password for each repository, the first time you wish to use it.

- From the "Window" menu, select "Preferences".

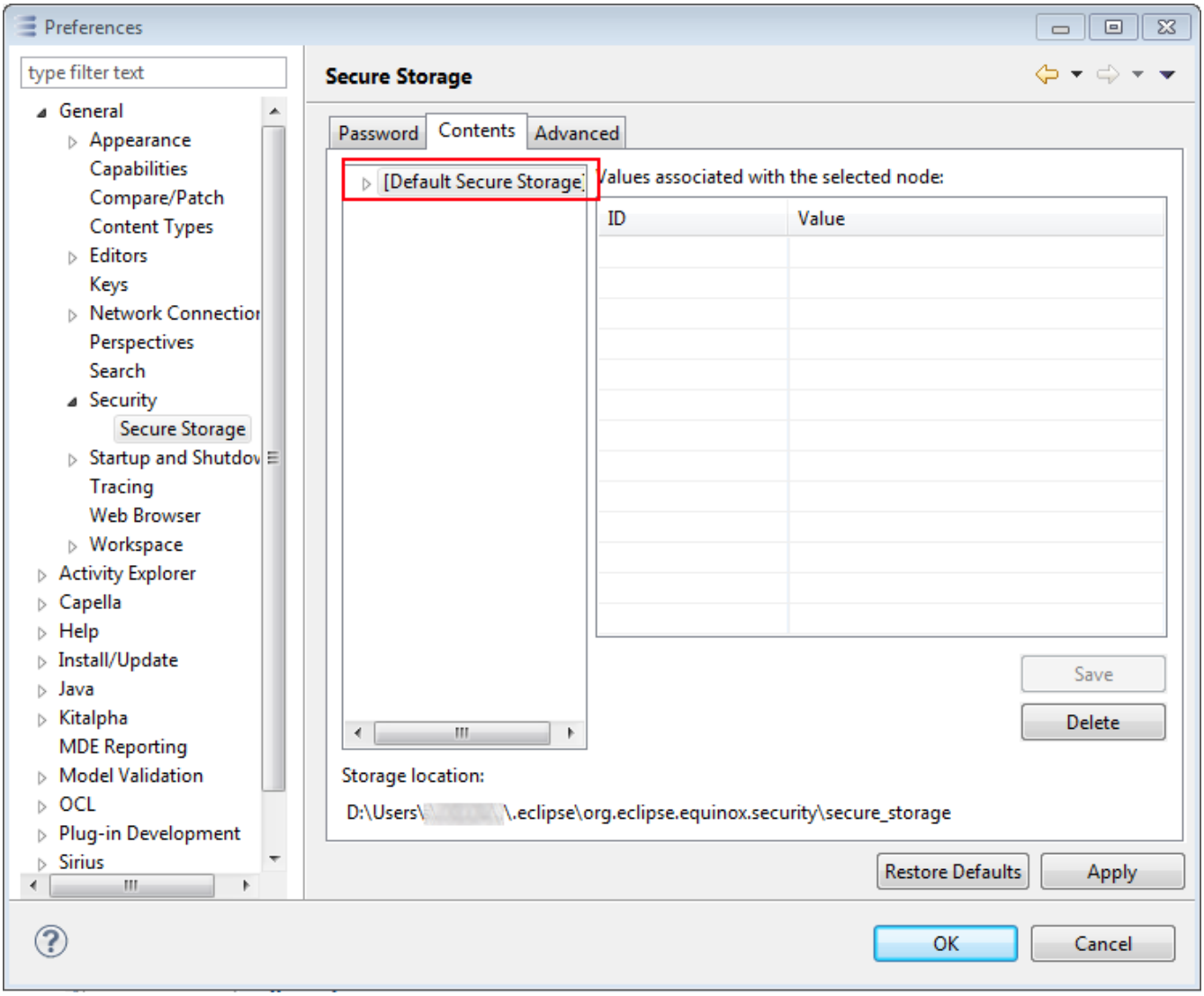


- Within the tree structure on the left-hand side of the "Preferences" window, open up the "General" entry and then subsequently the "Security" entry. Select the entry "Secure Storage".

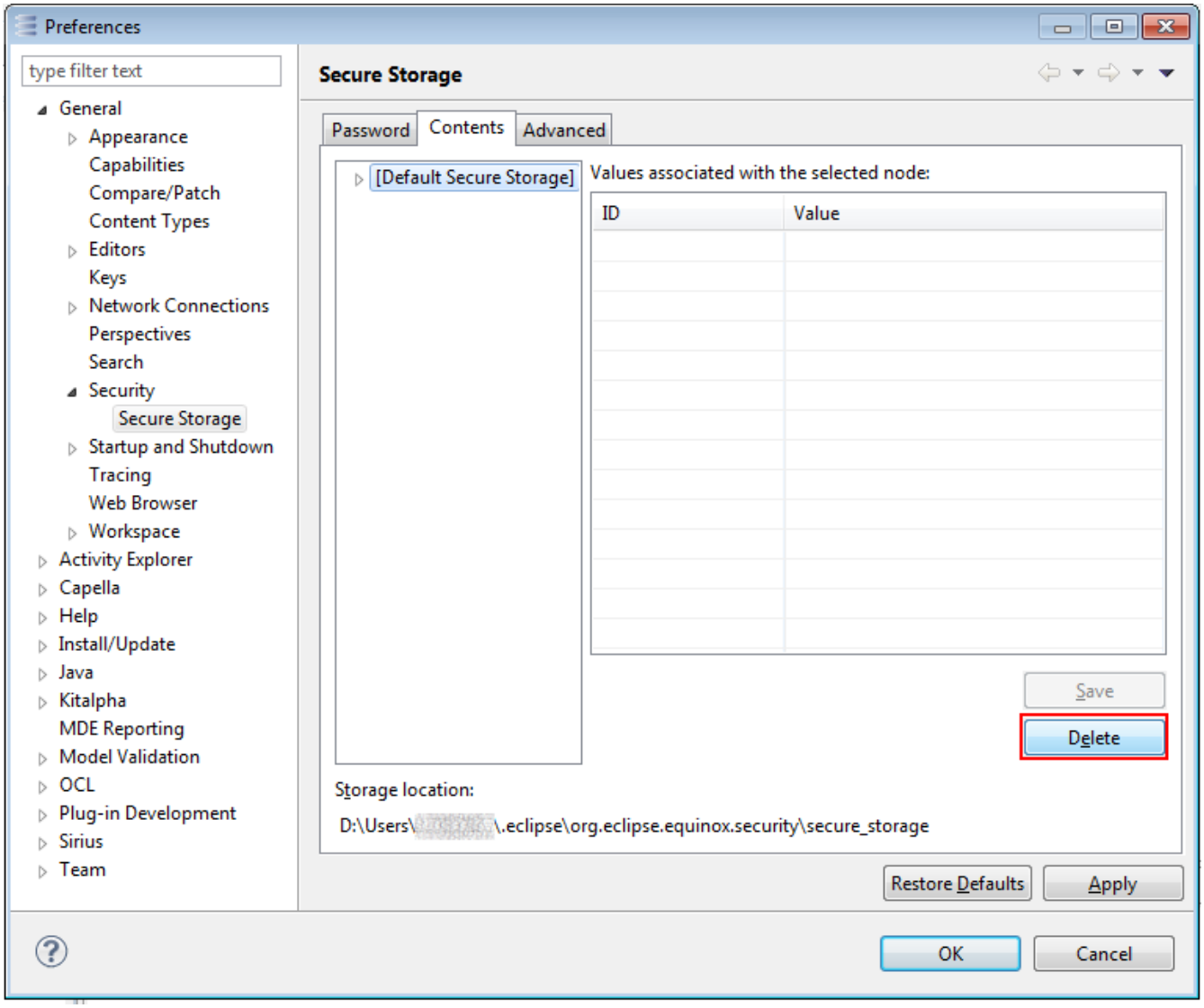


- In the right-hand panel of the "Preferences" window, select the "Contents" tab and then the

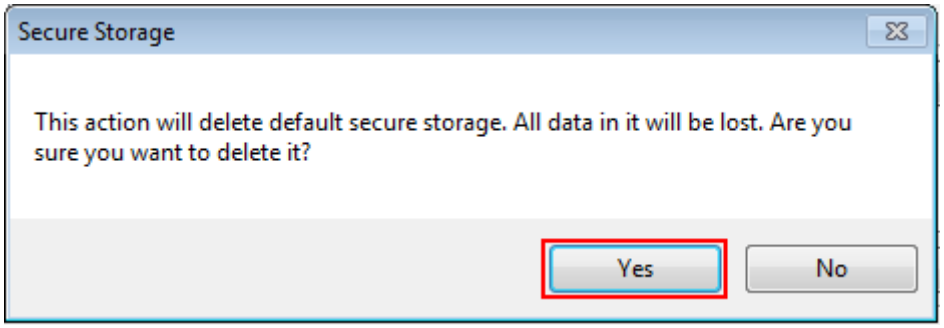
entry "[Default Secure Storage]".



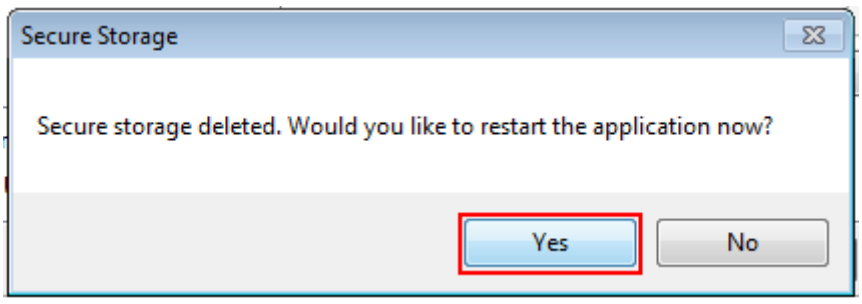
- Press the "Delete" button.



- When asked if you wish to delete the password store, select "Yes".



- You will then be prompted to restart Team for Capella. Select "Yes" and wait until the application restarts.



3.4. Aird Fragments Connection

Introduction

The purpose of this functionality is to be able to connect to airdfragments to work with the **whole semantic model but only a subset of representations** (diagrams or tables).

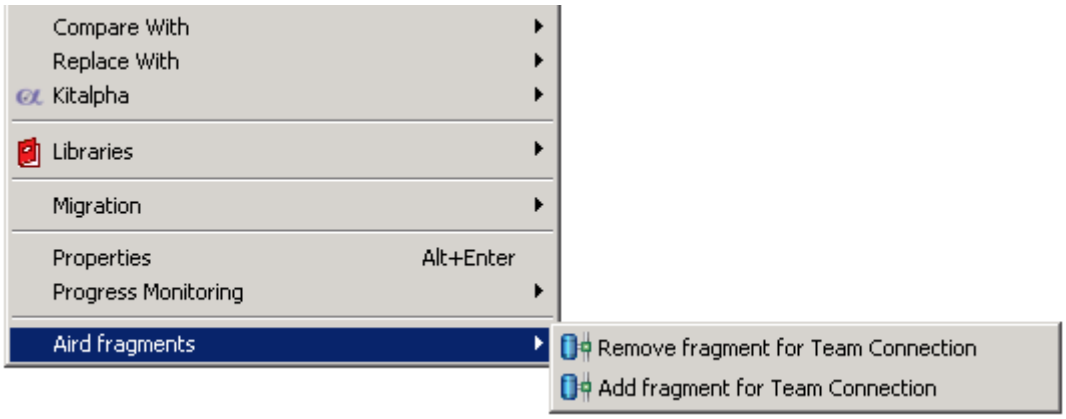
It can be useful when working with a big model to shorten connection time and memory consumption.

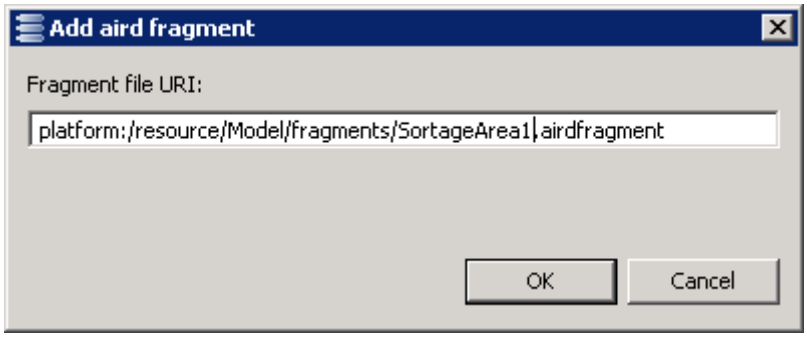
Model Preparation

The model to prepare must be a **local model in file format** (do an import if necessary). **The session must be open**.

Two actions can be used:

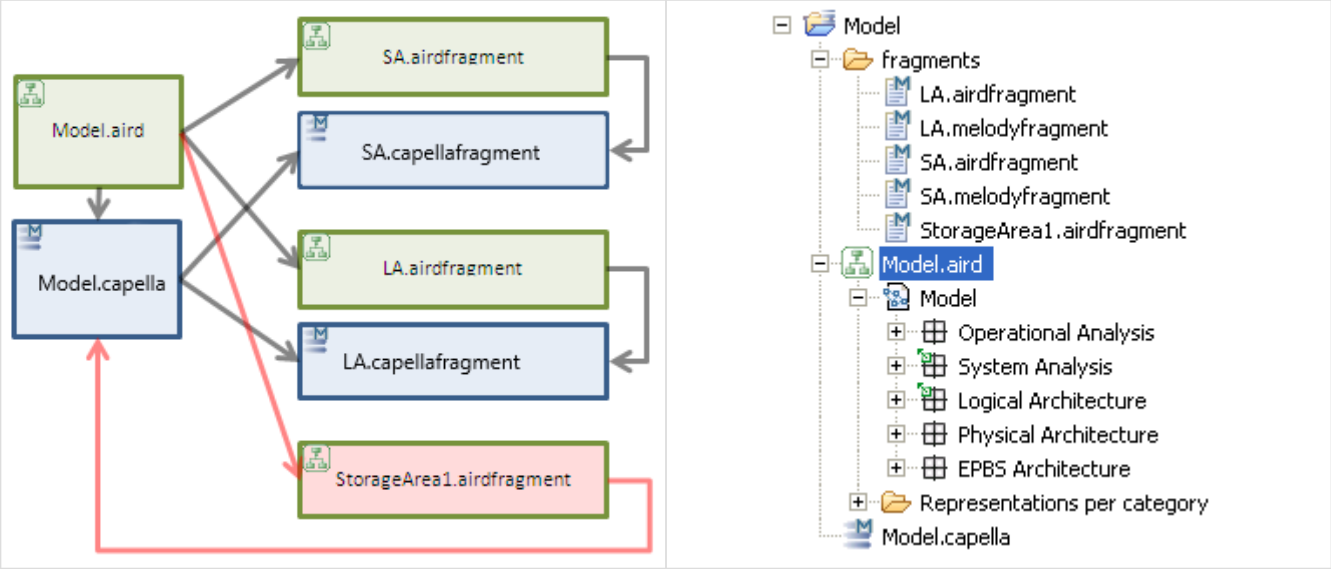
- "Fragments...": allows creating classic Capella fragments (see the Capella Model Configuration Management Guide in the Capella User Manual). This action is called on a semantic element and creates 2 files:
  - A .airfragment: containing diagrams,
  - A .melodyfragment: containing semantic elements,
- "Add fragment for Team Connection": allows creating airdfragments to store diagrams.
  - This action is available in the contextual menu of the .aird file:





It must be added in the project (in the project root or in a directory of the project, "fragments", for example).

Model organization after an execution of this action:



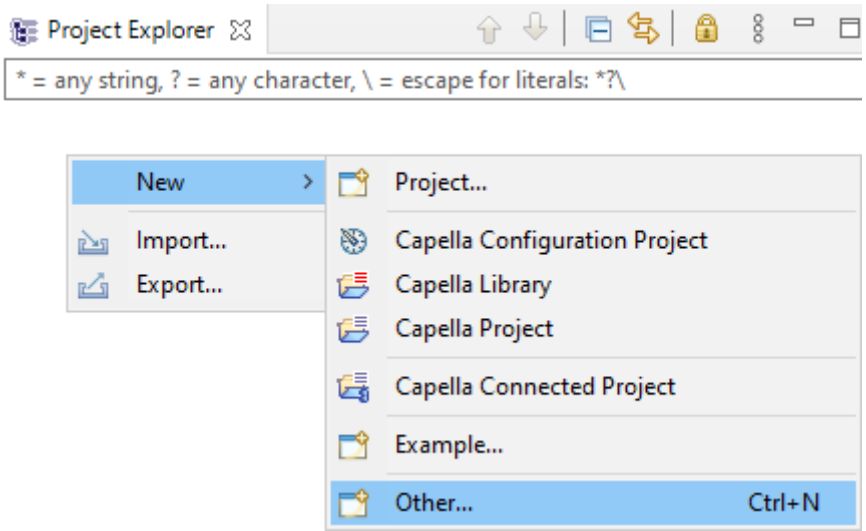
Restrictions

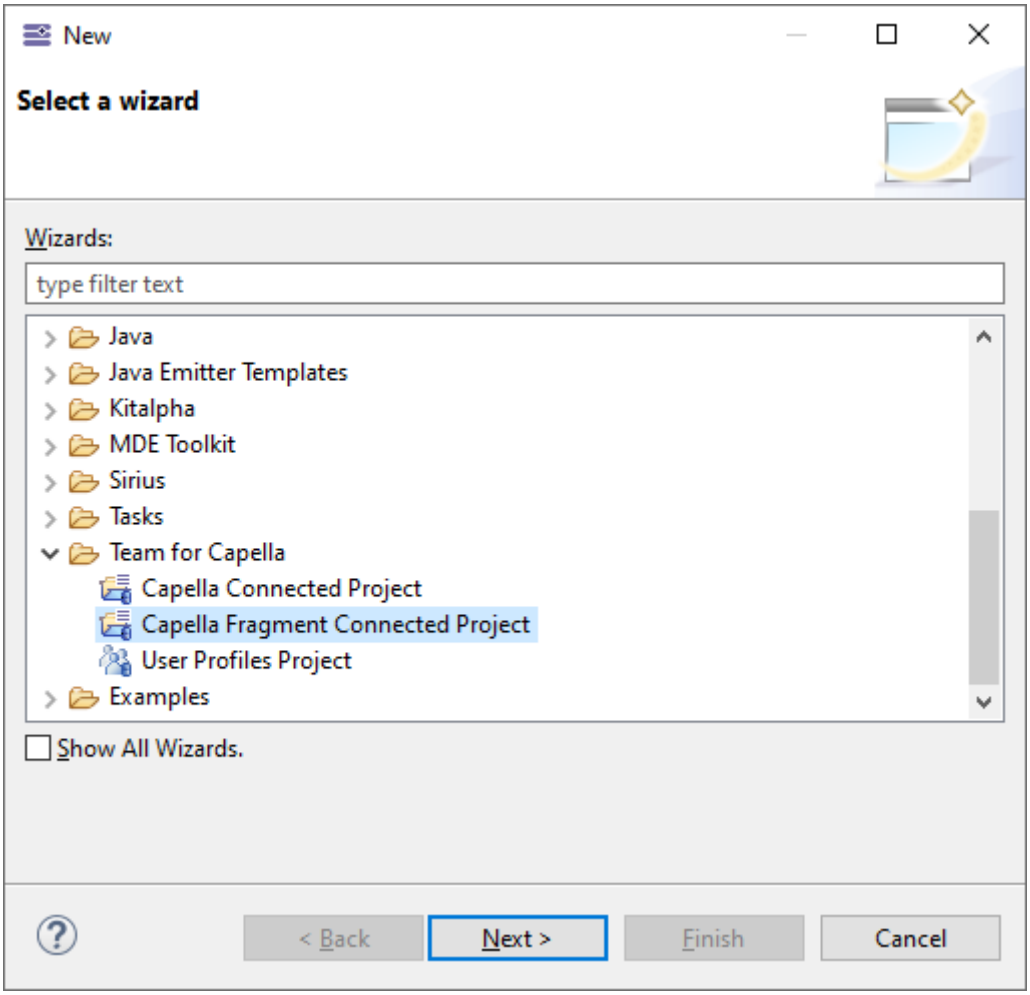
- The .airdfragment file path must not contain spaces.
- The project containing the airdfragments must not host many semantic models (only one semantic model is allowed).

Connect to Airdfragments

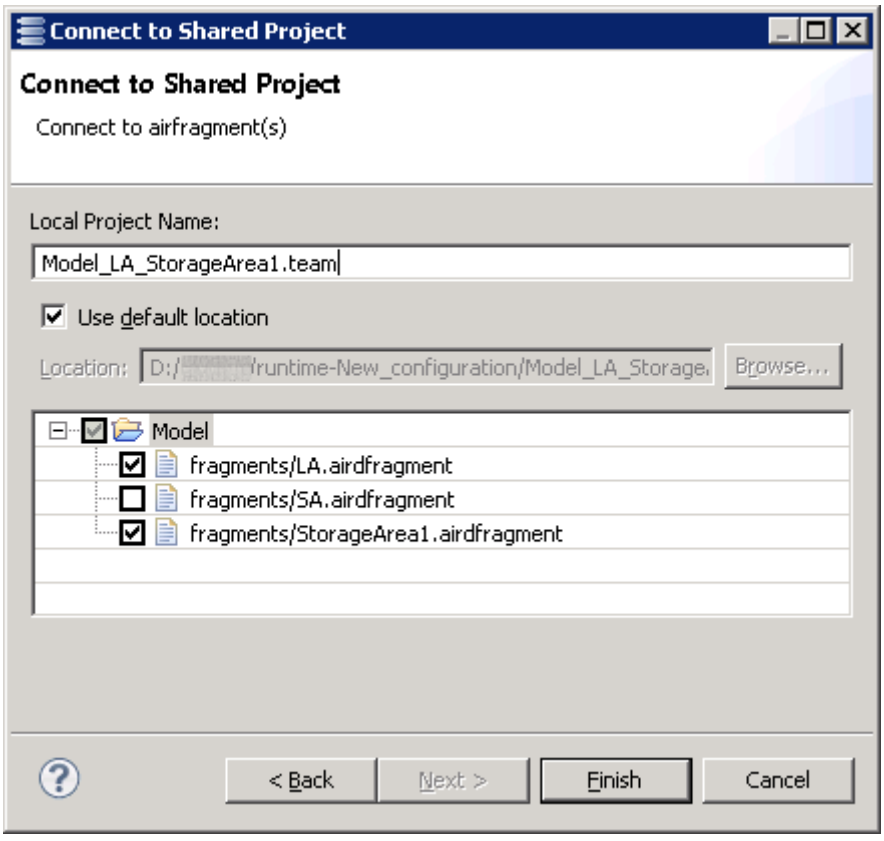
When the model is well organized, export it to the server.

You can create connection projects to several .airdfragments thanks to the dedicated wizard:





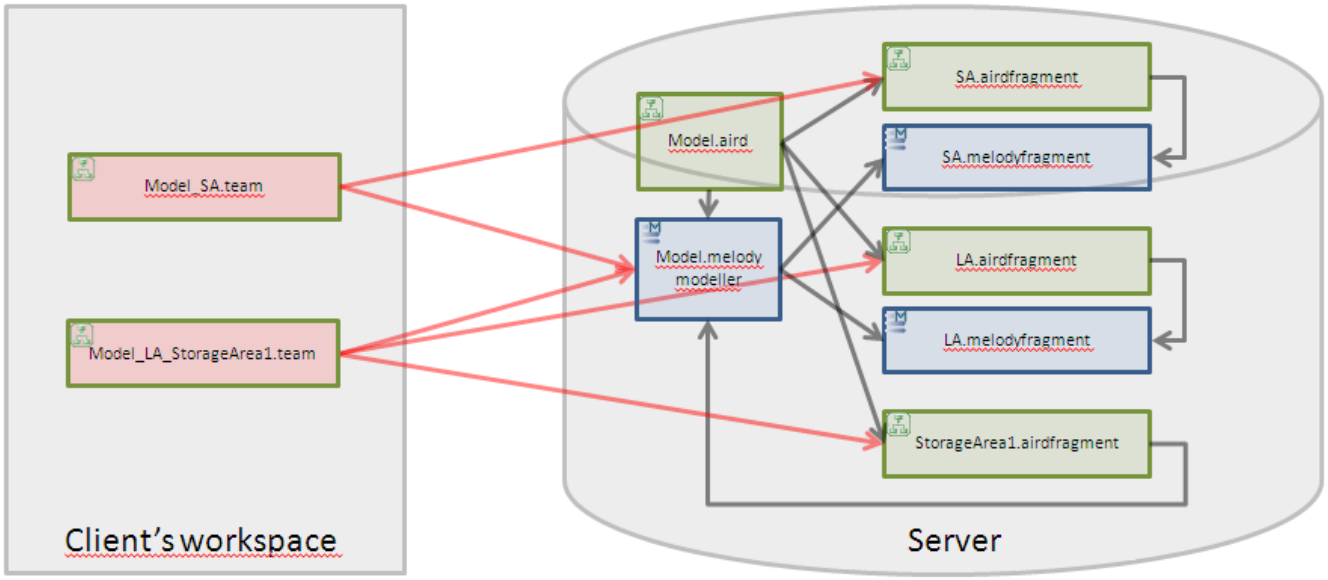
The second page of the connection wizard allows selecting .airfragments to use.



Connection to fragments belonging to different models is not allowed since it does

not make sense.

Connections to fragments example:



- A client using Model_SA.team will see:
  - The whole semantic model,
  - Diagrams contained in SA.airdfragment,
- A client using Model_LA_StorageArea1.team will see:
  - The whole semantic model,
  - Diagrams contained in LA.airdfragment and in StorageArea1.airdfragment.

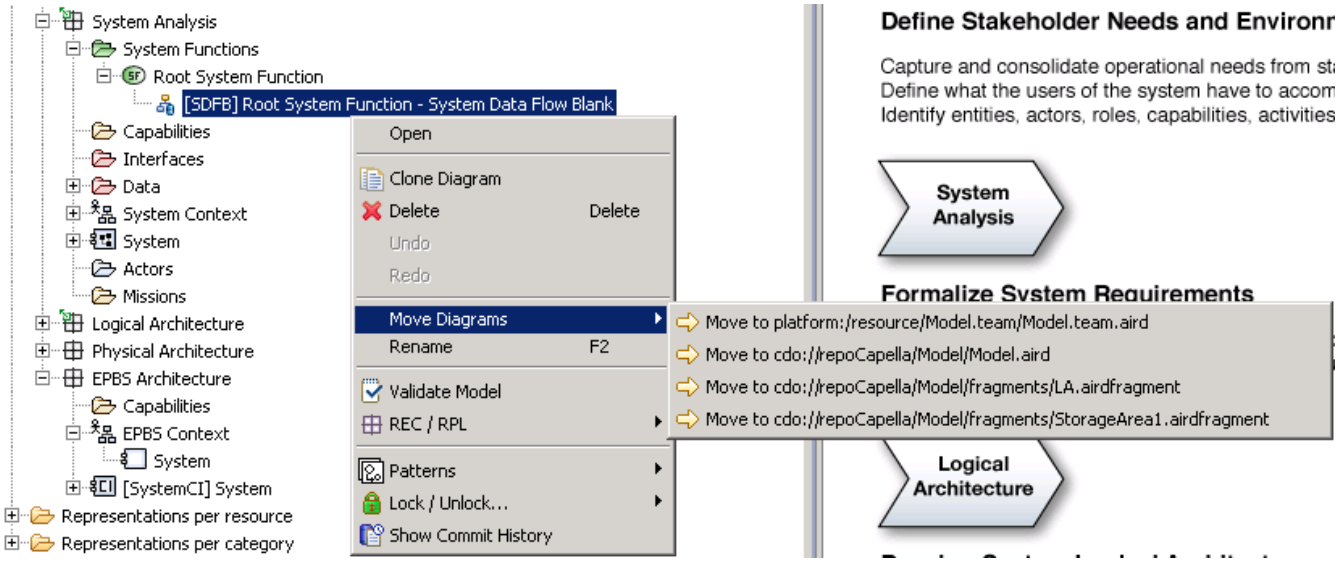
As previously, it is still possible to connect to the .aird, all diagrams will be accessible.

Diagrams Moving

It can be needed to move diagrams between aird and airdfragments and between 2 airdfragments.

This can be done on a local model or on a remote model (the source and destination resources must be visible from the same connection project).

To move a diagram to another resource, use the "Move Diagrams" sub menu:



In addition, to ease diagrams management, the "Representations per resource" item can be useful. To display it, uncheck it in the "Customize View..." dialog.

Airdfragments Management

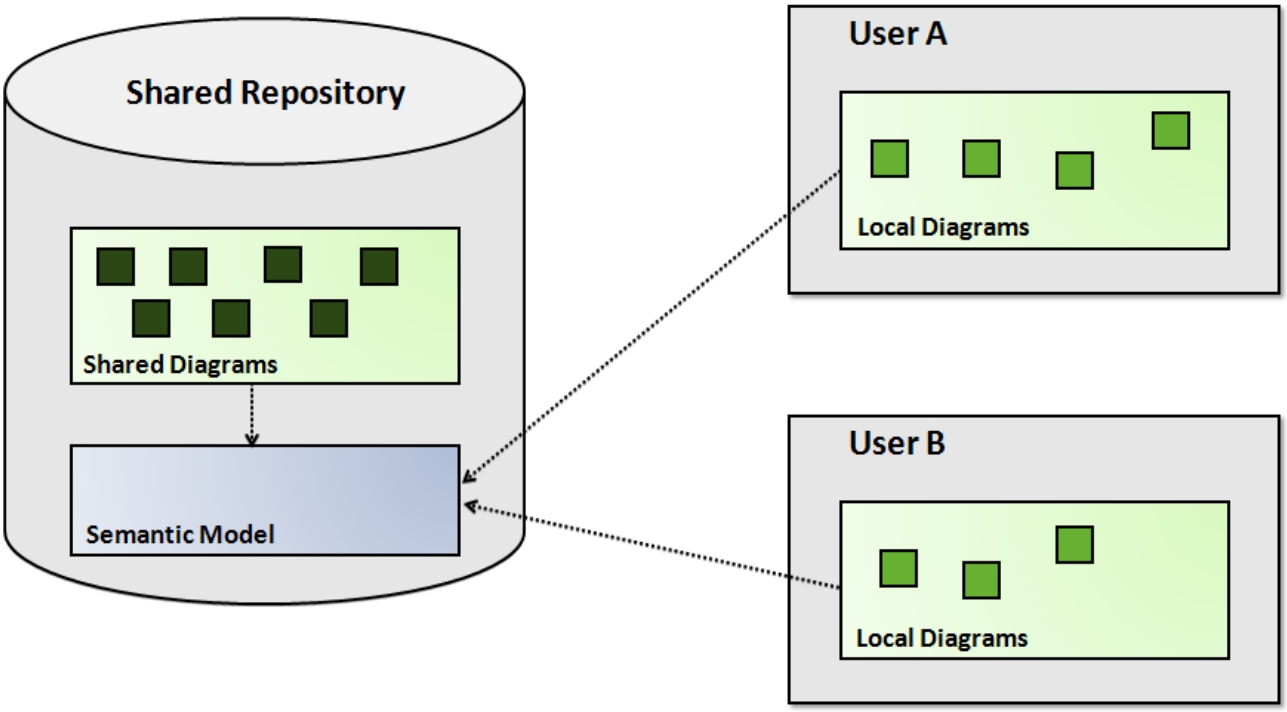
airdfragments can only be managed in a **local model** (do an import if needed).

- To get rid of classic Capella Fragments, use the "UnFragment..." command (see the Capella Guide/User Manual/Fragment management/Unfragmentation Command section in the Capella User Manual),
- To get rid of an airdfragment added with the "Remove fragments for Team Connection" command:
- Move all its contained diagrams to the .aird or another .airdfragment,
- Use the "Remove fragment for Team Connection" command,
- Manually delete the .airdfragment from the project.

Do not directly use the Eclipse delete command, all content would be lost.

3.5. Working on a Remote Model

Several users access the model held by the Team for Capella Server repository through their Team for Capella Client. The Capella project on the client side only consists in one ".aird" file which is both a proxy towards the shared repository and a container for the local diagrams.



Fundamental principles

- The **semantic model is always integrally shared**
- **Representations (Diagrams, Tables, Trees) can be shared** in the repository **or can be local** to one user
- **Locks are taken automatically** as soon as an element or a representation is modified.
- When a user has a lock (displayed with a green lock decoration), he can edit the element (rename an attribute, add/remove sub-elements). The other users cannot edit this element (displayed with a red lock decoration).
- **Locks are automatically released when committing**.
- By default, **any Save action triggers a commit**.
- It is possible for a user to set explicit locks (i.e., force the lock of an element or set of elements before modification). Explicit locks are not released when saving the modifications. The elements stay locked until the user explicitly unlocks them.
- From a diagram editor, modifying an element property visible on the diagram will lock the diagram.
- Locking a diagram does not lock the semantic elements presented on this diagram.
- **Locking a diagram prevents others from modifying this diagram, but does not prevent other users from modifying non-locked semantic elements represented on this diagram**.
- Adding an element A in an element B requires a lock on both A and B

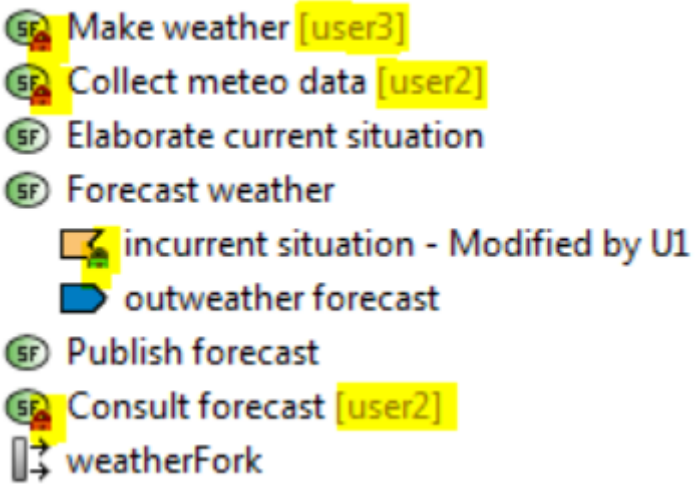
- Newly created elements are not locked

Locks and Update on Model Elements

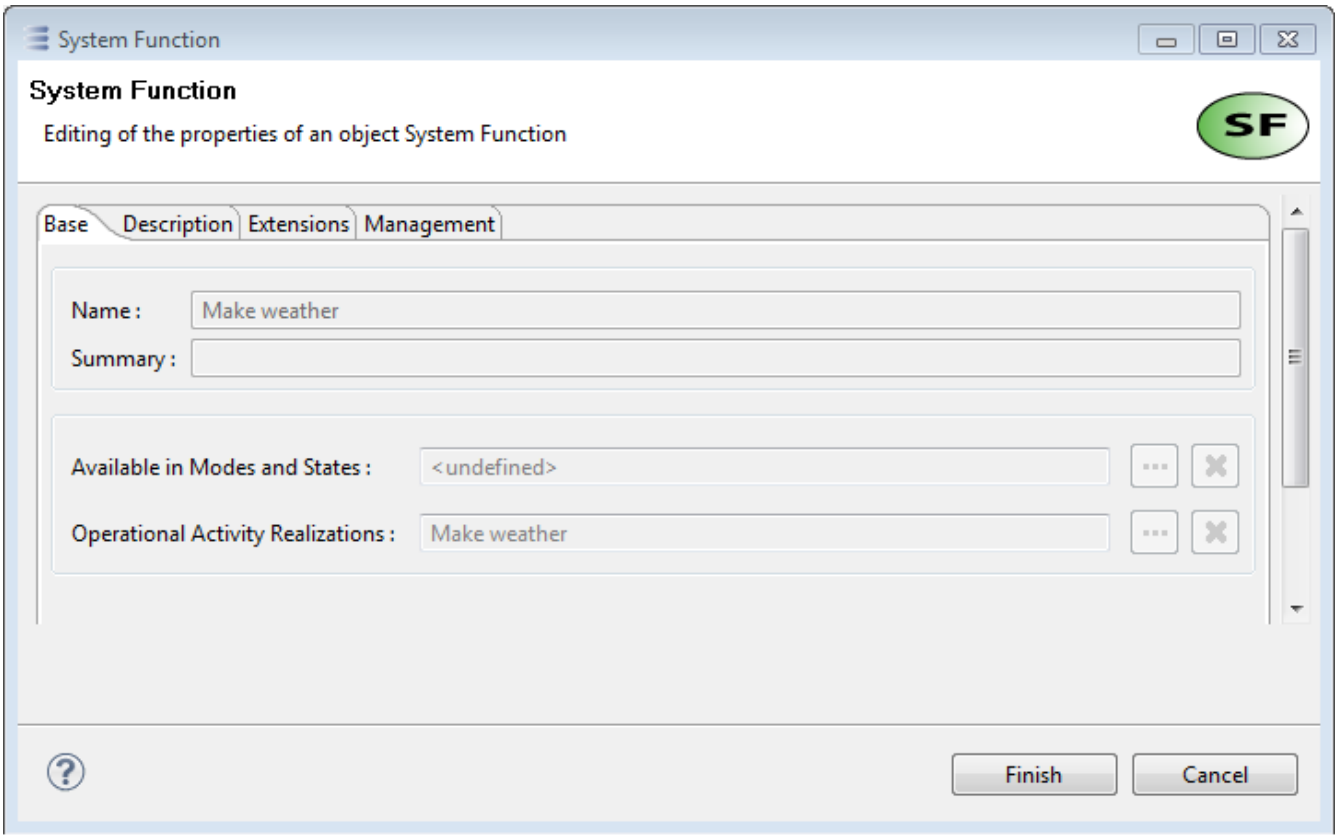
Red locks indicate another user is currently modifying the element (this modification might be a deletion). The identification of the user holding the lock is added between brackets as a suffix.

Green locks indicate the current user has reserved or modified the current element.

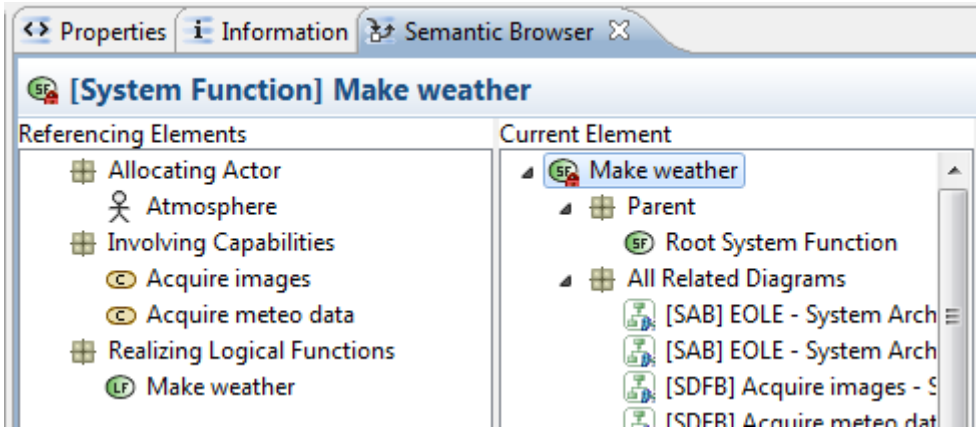
Below is an example of the decorations in the Project Explorer.

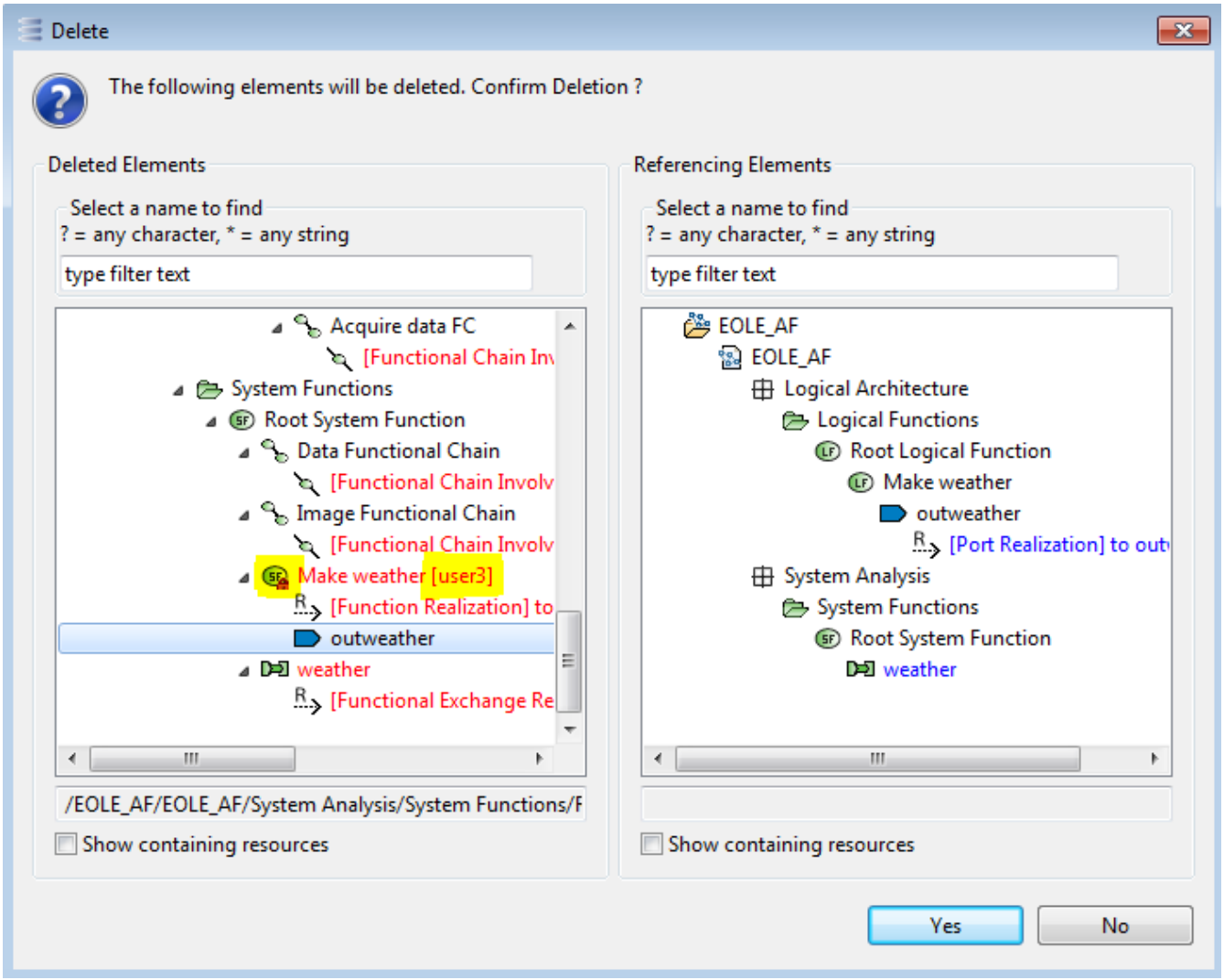


When an element is locked by another user, its editor dialog is still accessible but cannot be modified (all fields are disabled).

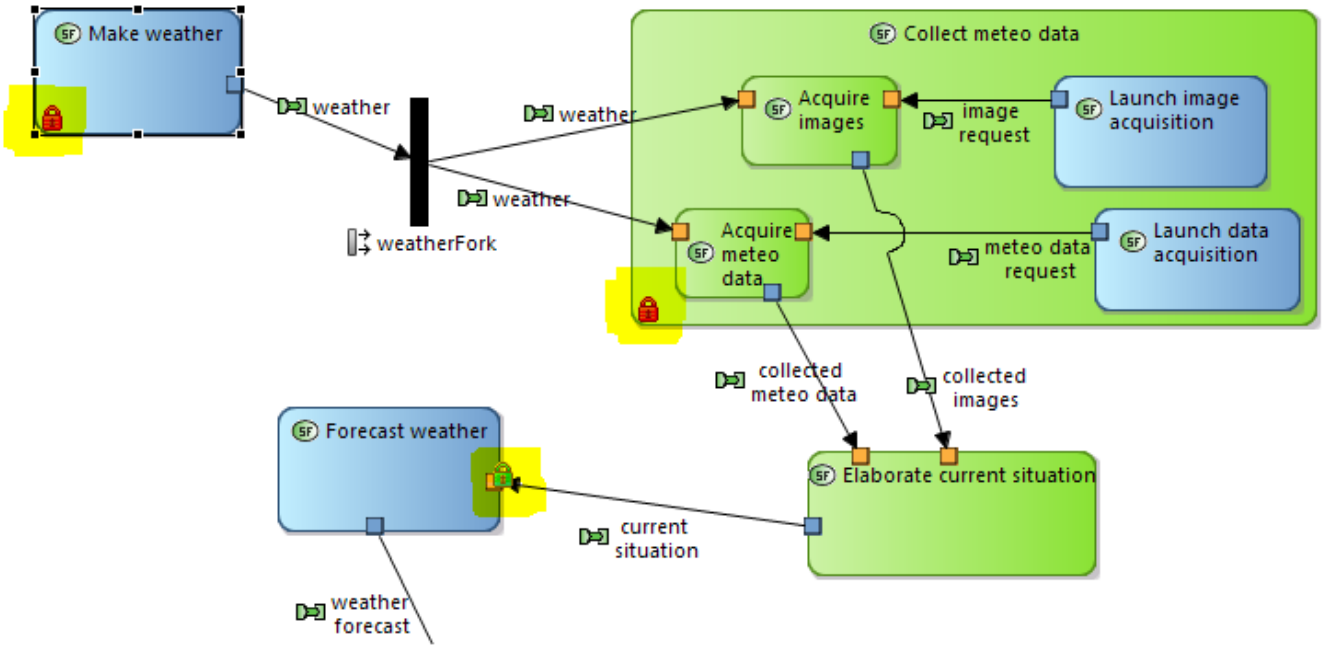


Lock decorations are visible in any View of Capella, such as the Semantic Browser, the selection dialogs or the delete confirmation window.





On diagrams, the semantic locks are represented on the graphical artifacts (containers, nodes, ports, links) representing the locked model elements.



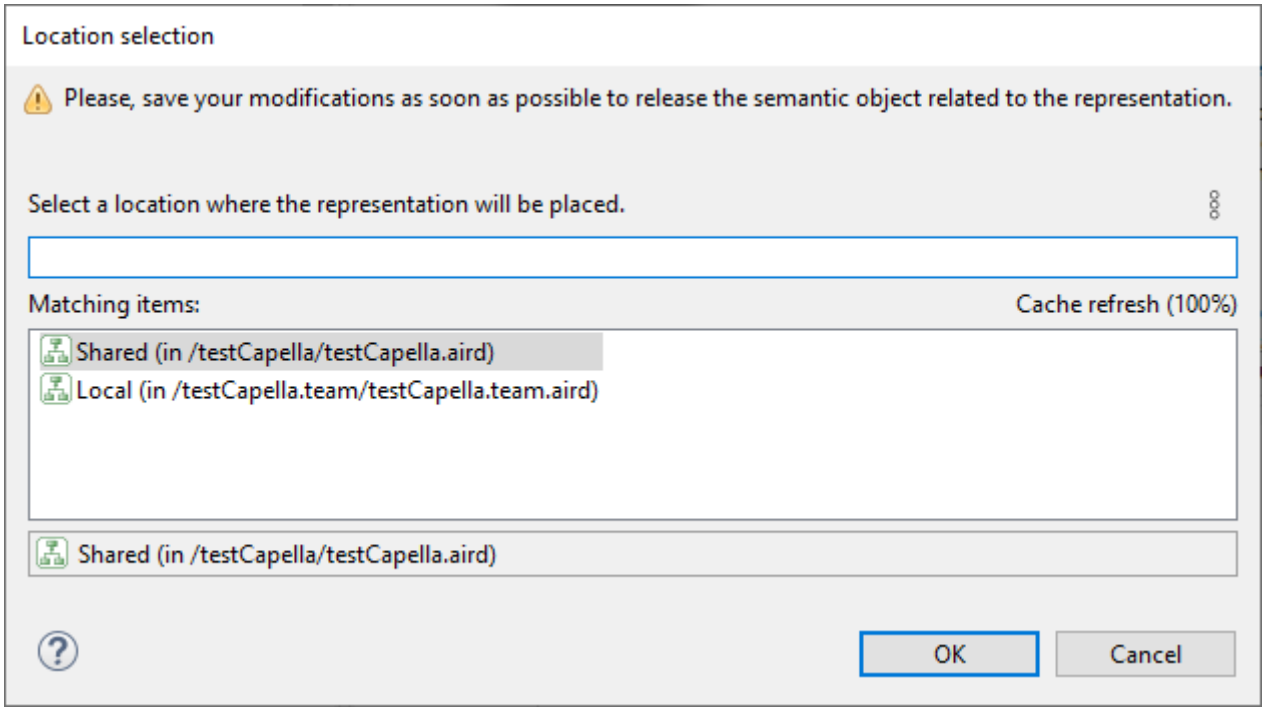
Updates of modified semantic elements are performed automatically.

Locks and Updates on Diagrams

Two users cannot work simultaneously on the same diagram. As soon as a user modifies a diagram, the whole diagram is locked for the other users.

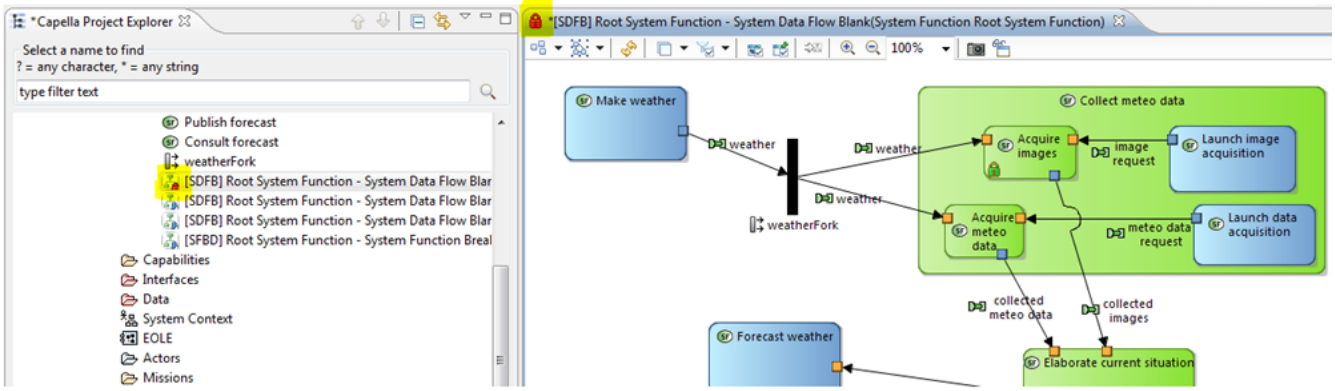
When creating, cloning or moving a representation, **the associated semantic target element is automatically locked**. This is useful to avoid that, on a connected project, the current user saves the newly created representation with a null target in case another user had deleted the target just before the current user saves. Note that a warning is displayed in the dialog box to ask the user to save as soon as possible so that to release the lock.

This behavior can be deactivated using the preference **CDOSiriusPreferenceKeys.PREF_LOCK_SEMANTIC_TARGET_AT_REPRESENTATION_LOCATION_CHANGE** with a false value.



This behavior has a particular impact when using User Profile. If the user has only a read only right on the semantic element, he cannot create/clone/move a representation on it.

The lock diagram decorations are visible both on the tab bar of the diagram editor and in the Project Explorer.



When a diagram is locked by another user:

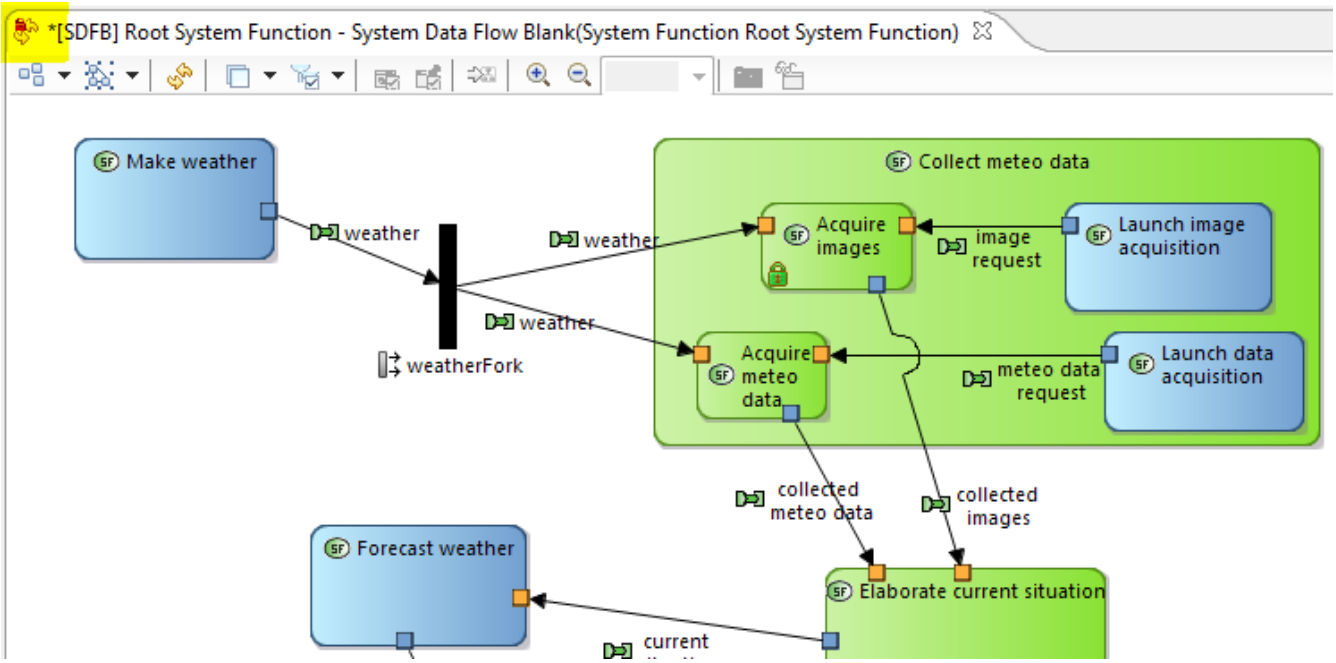
- Moving or resizing elements is not possible
- Changing the colors of elements is not possible
- Adding or removing elements is not possible
- Changing the label of an element is not possible

However, **even though another user locks a diagram, semantic elements appearing on this diagram can still be modified by anyone**. This is the case for example of the Function "Acquire Images" on the above example. The opposite is true as well: one can have a green lock on a diagram despite some semantic elements appearing on this diagram are locked by other users.

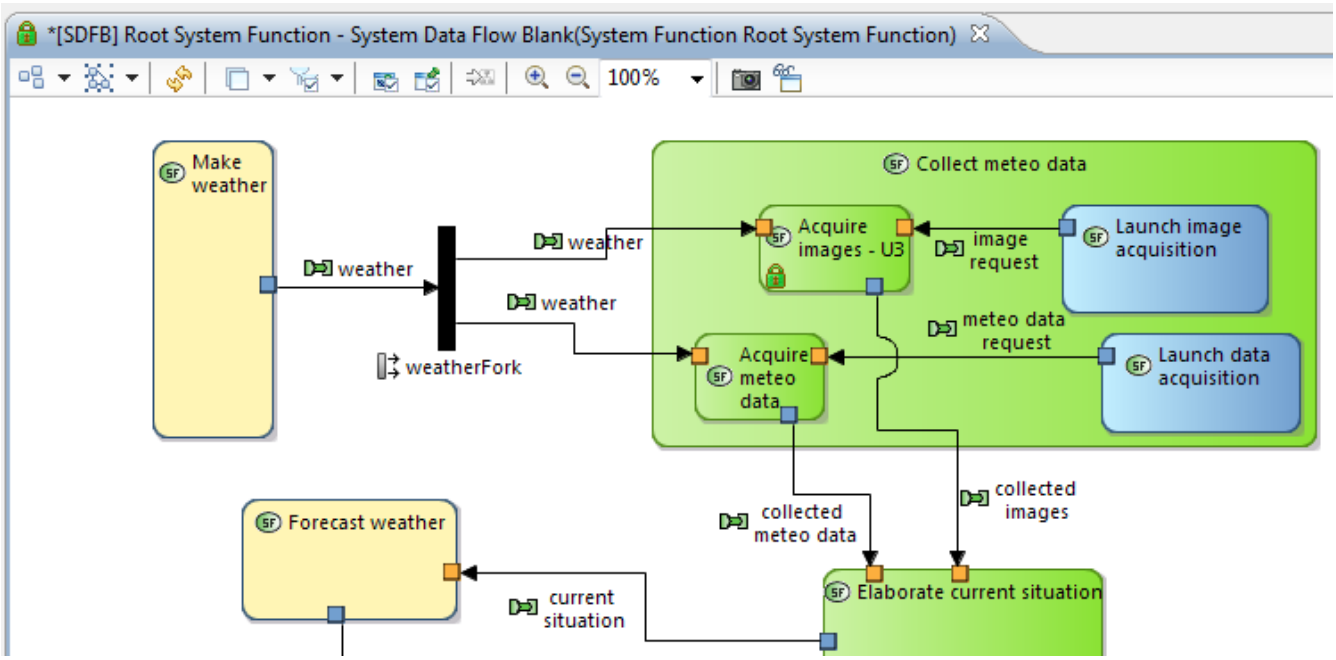
Once the user modifying a diagram saves and commits its modifications, the diagram is not locked

anymore. For the other users currently displaying the diagrams, two different alternatives:

- **If the refresh strategy is "automatic", then the diagram is instantly refreshed.** For performance reasons, this alternative is not recommended.
- **If the refresh strategy is set to "manual", then a specific decorator indicates the diagram needs to be updated**.



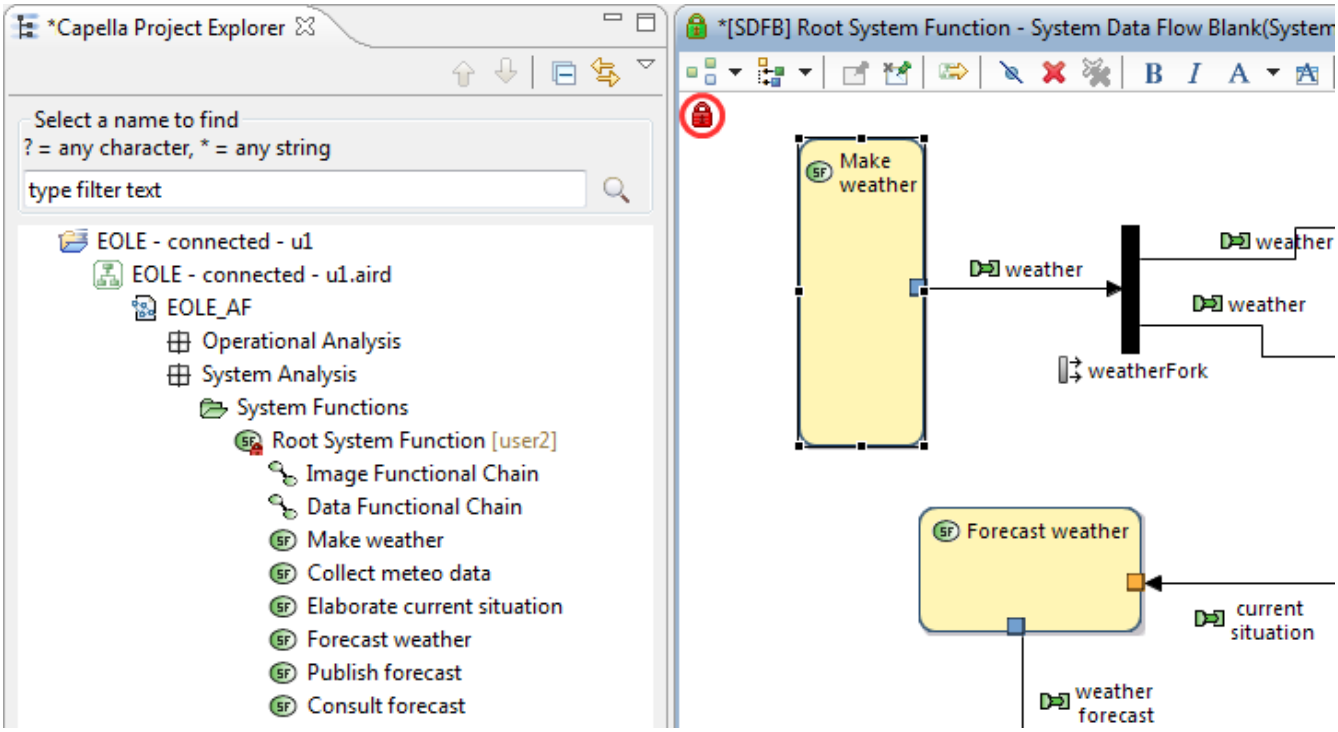
After the refresh is performed, the new layout becomes visible.



Note: on the above example, one semantic element ("Acquire Images") was currently being renamed by the user. The consequence is that the refresh induces a new change (and thus a green lock) on the diagram to reflect the label update.

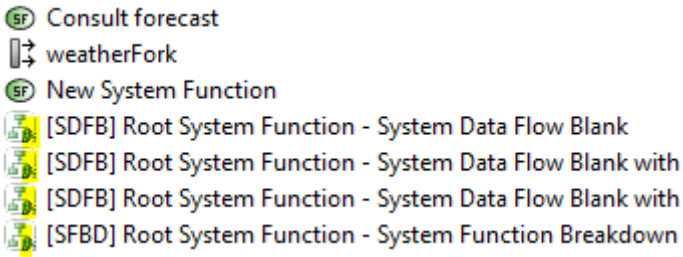
In Capella, the background of diagrams always represents a semantic element (which is the element under which the diagram is located in the Project Explorer). In case this semantic element is locked (hereunder the Root System Function), a specific decorator is put on the background of the

diagram. This means, for example, that even though the diagram is locked for edition (green lock), adding a new element on the background of the diagram is not possible.

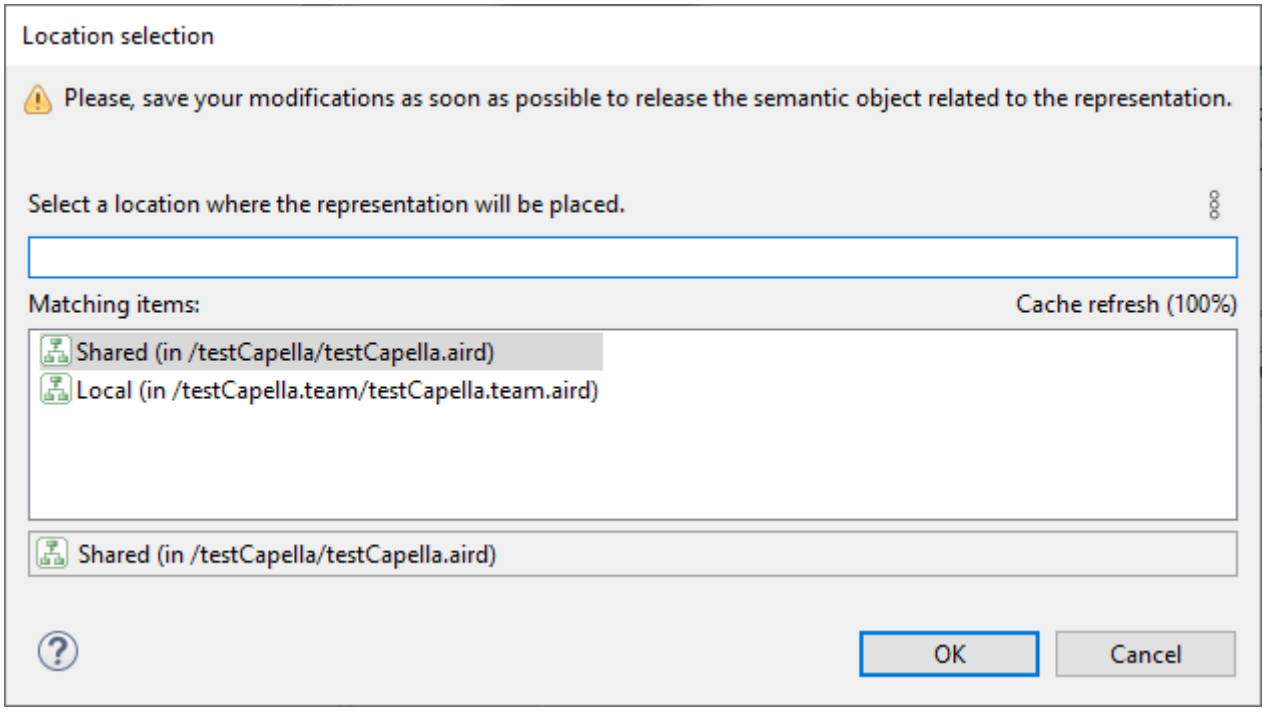


Local vs Shared Diagrams

Diagrams can be local or shared in the repository. Shared diagrams have specific decorators.

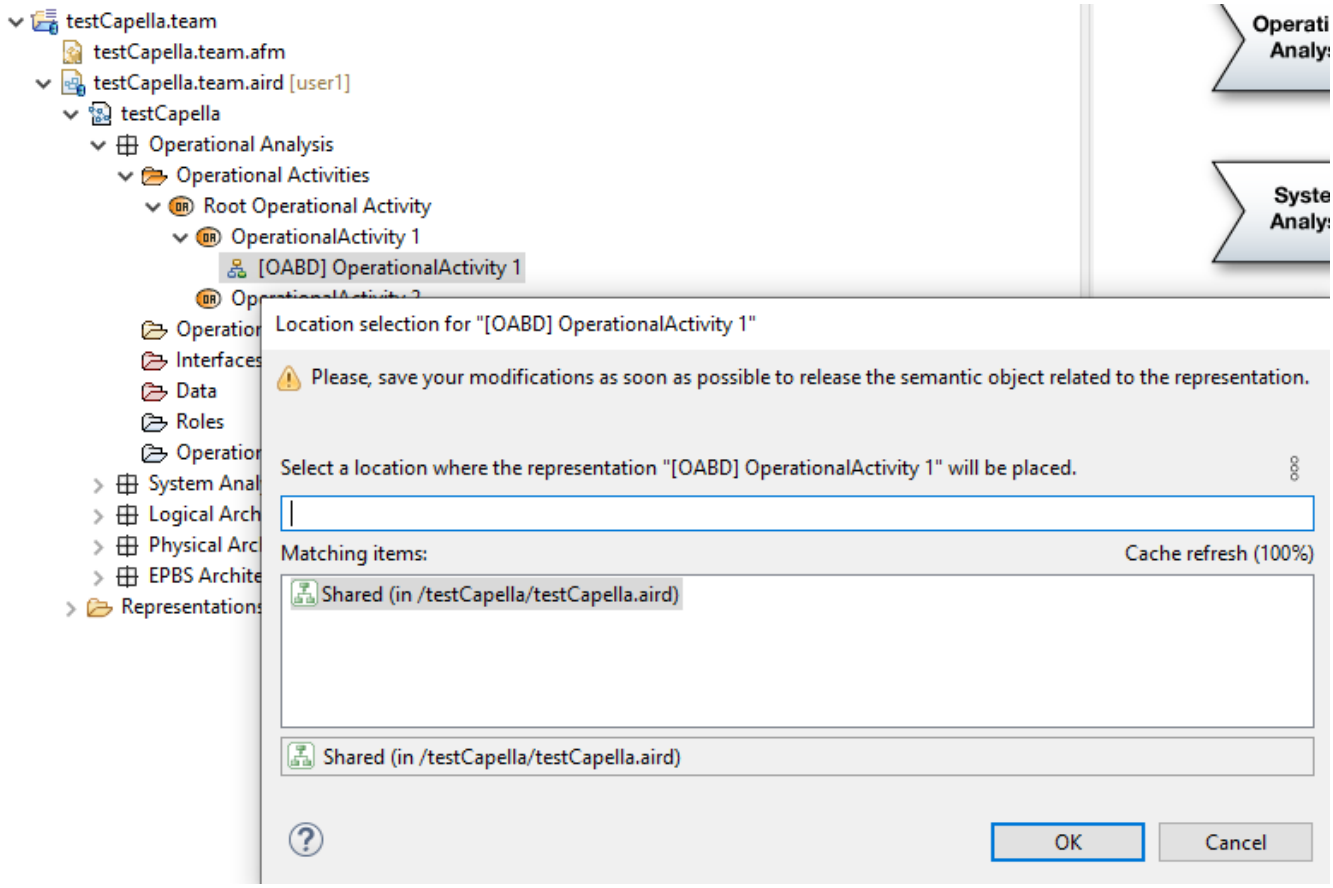


When creating a new diagram, a dialog pops up asking the user to choose whether the diagram should be shared (cdo://) or local (platform:/resource...).

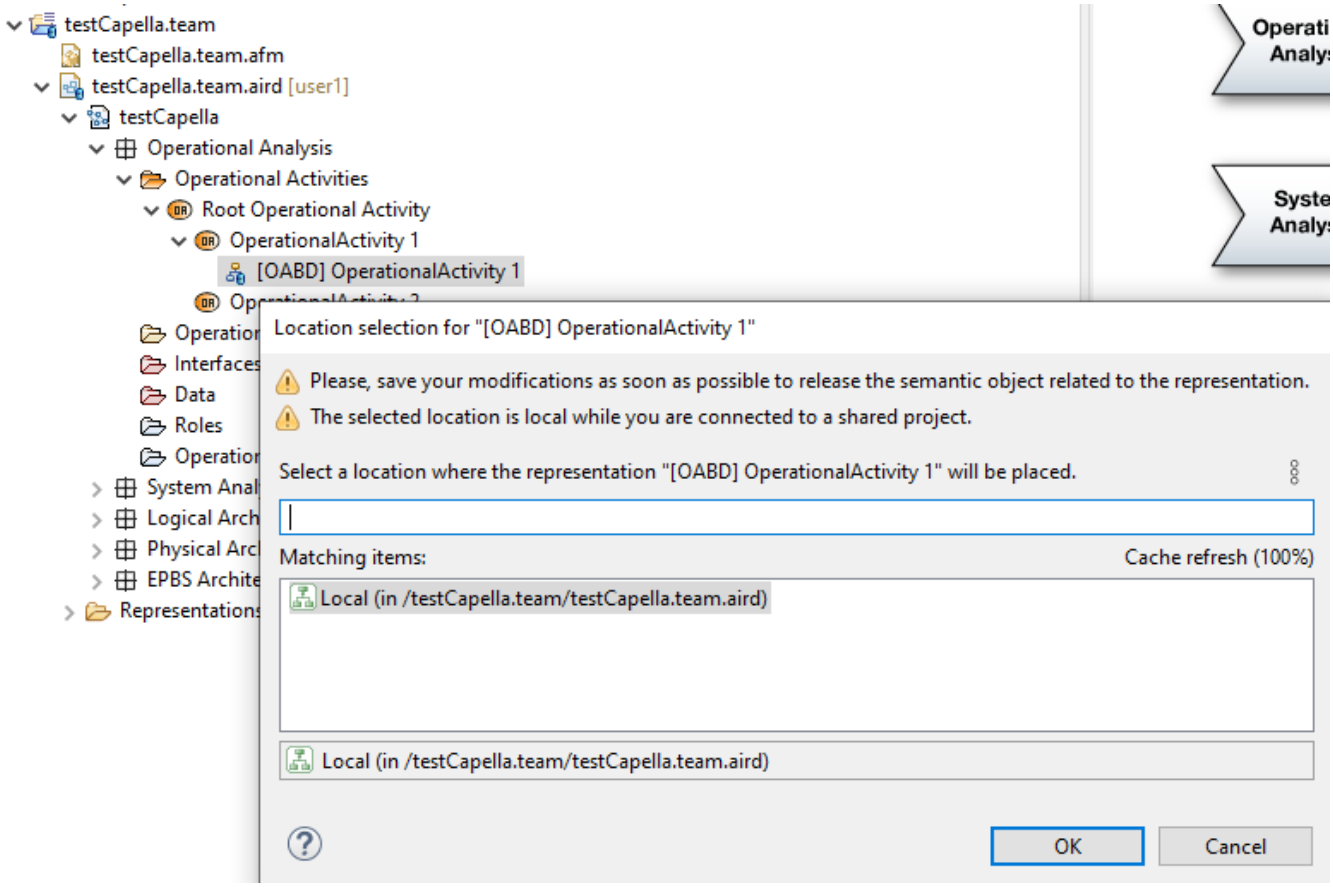


It is possible to move diagrams from the repository to the local project and vice versa.

From the local project to the shared repository.



From the repository to the local project.

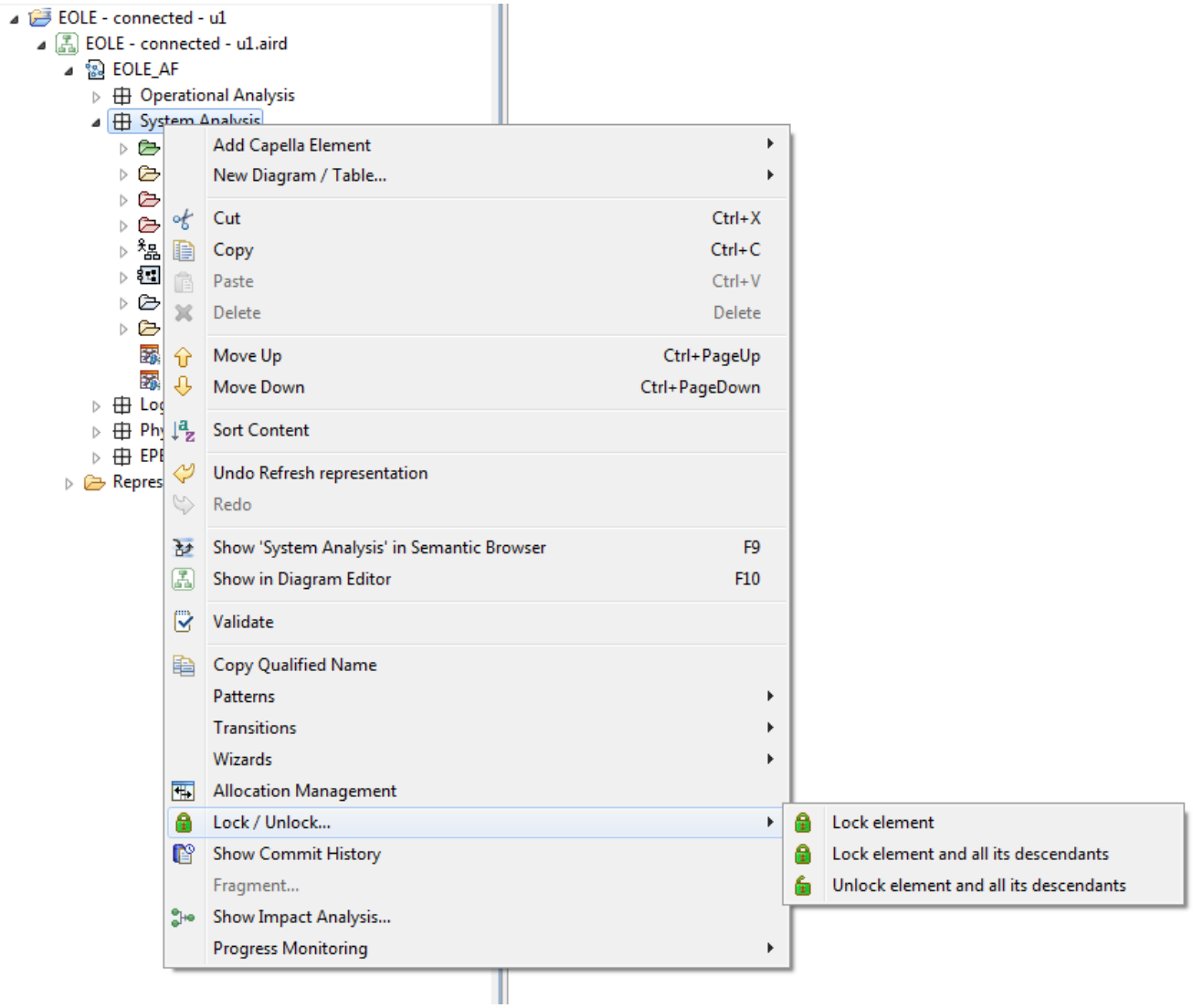


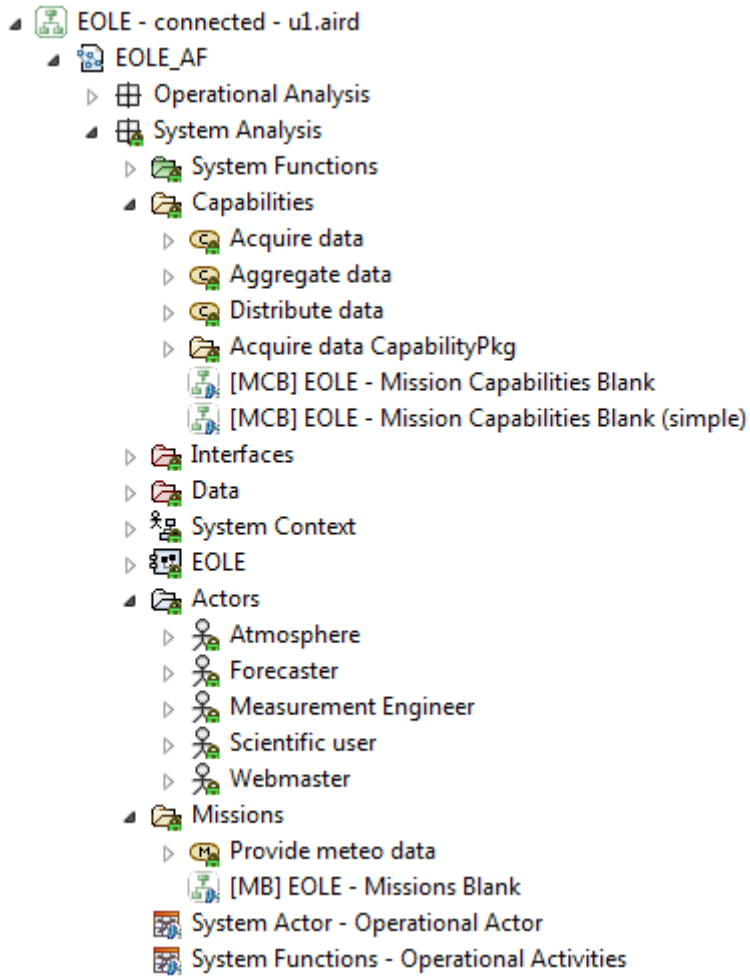
Note that there is a warning when the selected target is local.

Semantic elements created on a local diagram are instantaneously shared with other users as soon as a commit is performed. Local diagram does not mean local elements.

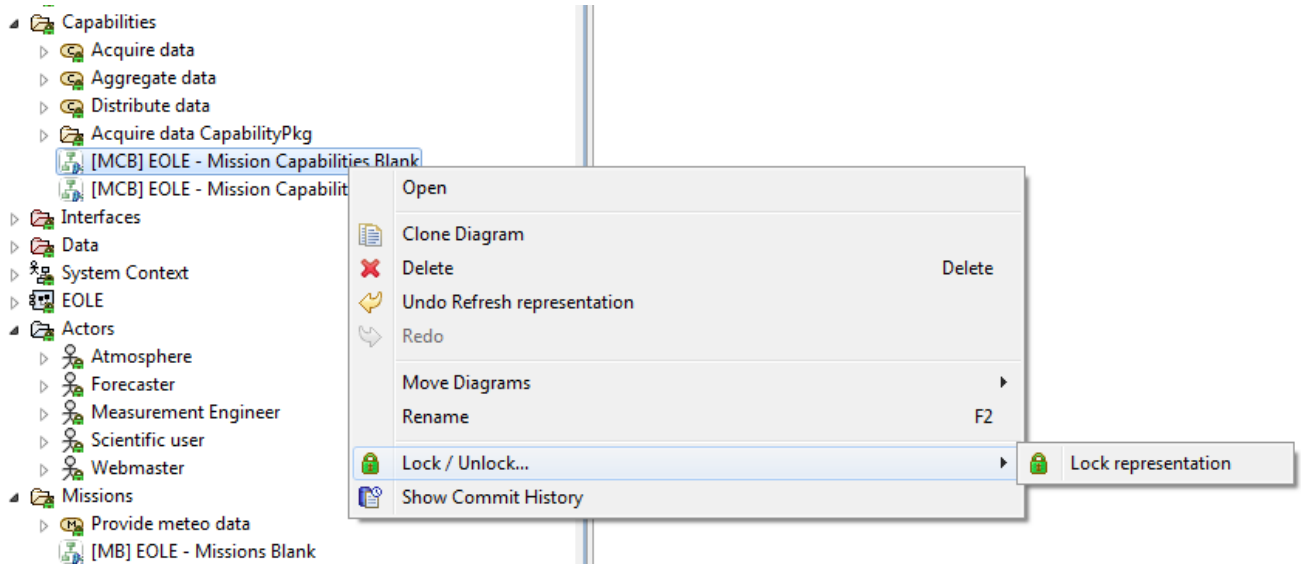
Explicit Locks

It is possible to explicitly lock an (or a set of) element(s) by using the contextual menu.


EOLE - connected - u1
EOLE - connected - u1.aird
EOLE_AF
Operational Analysis
System Analysis
Add Capella Element
New Diagram / Table...
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Move Up Ctrl+PageUp
Move Down Ctrl+PageDown
Sort Content
Undo Refresh representation
Redo
Show 'System Analysis' in Semantic Browser F9
Show in Diagram Editor F10
Validate
Copy Qualified Name
Patterns
Transitions
Wizards
Allocation Management
Lock / Unlock...
Show Commit History
Fragment...
Show Impact Analysis...
Progress Monitoring
Lock element
Lock element and all its descendants
Unlock element and all its descendants



Note that only semantic elements are locked. Diagrams can also be locked explicitly, but individually.



The behavior of the locks when they are set manually is a bit different from the one of automated locks: while automated locks are systematically released at each commit, elements locked explicitly have to be unlocked explicitly as well.

Consider the following use case

- Element A and B are explicitly locked.

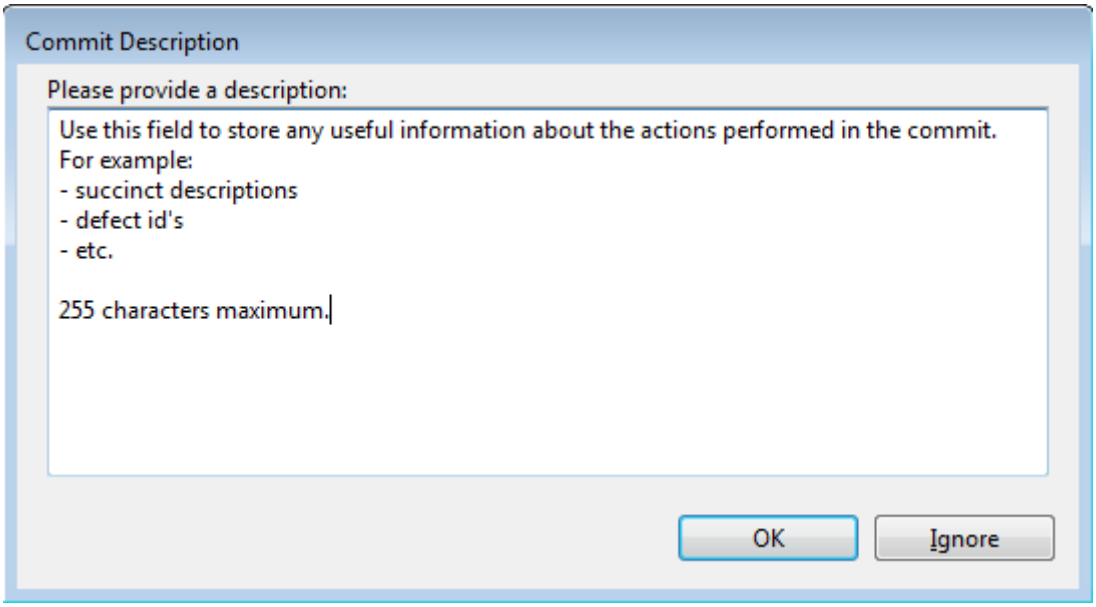
- Element C is automatically locked because modified.
- Element B is modified.
- A commit action is performed:
  - Lock on A is kept.
  - Lock on B is kept, but the modification on B is committed.
  - Lock on C is released and the modification on C is committed.

Dissociated local Saves and Commits

Currently not available.

Commit Descriptions and History

A Preference allows specifying whether a description is required when committing or not. In case this option is enabled, the following dialog is prompted on each commit action.



Dialog buttons:

- OK: the commit is performed with the given commit description.
- Ignore: the commit is performed without the commit description.
- Cancel: the commit is canceled. In this case, the user changes are kept unsaved and are still visible locally.

Another preference allows the user to pre-fill the commit description using various strategies. The default strategy exploits the previous commit description, while the Mylyn strategy relies on the content of the currently-active, non-completed Mylyn task using the template defined in the *Mylyn > Team* preferences. Below is an example of such a template:

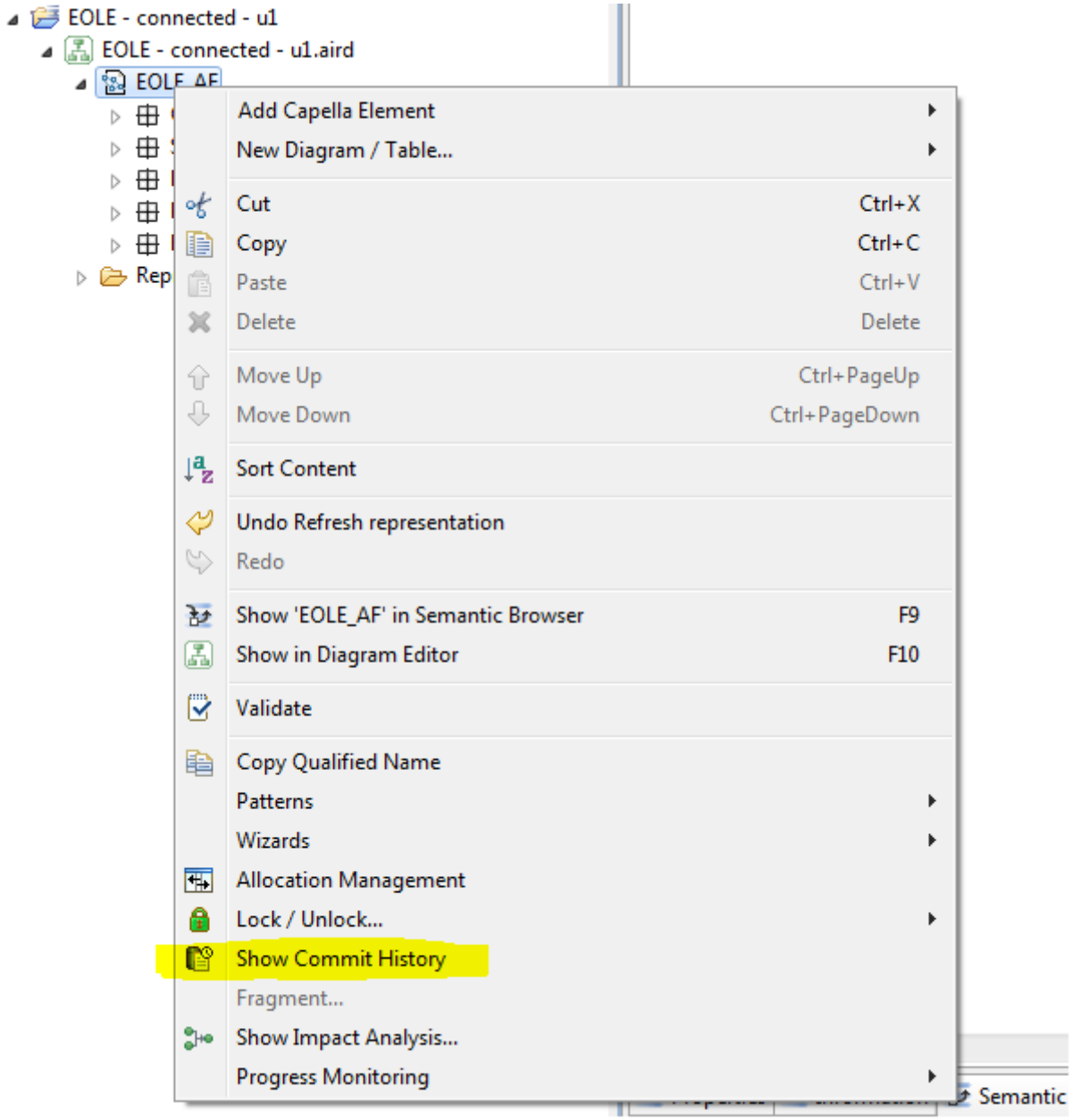
```
${task.description}
```

```
User Information:
```

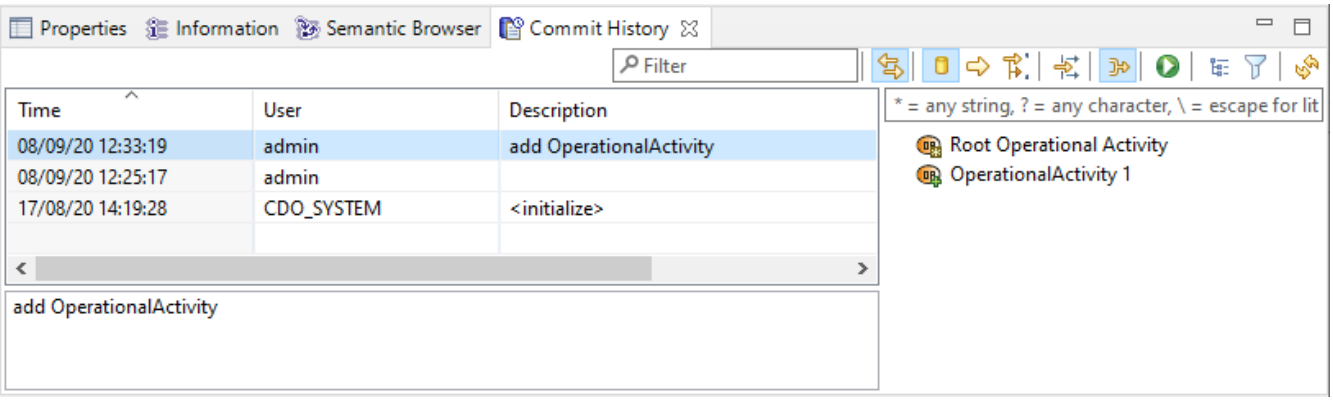
```
Key: ${task.key}
URL: ${task.url}
```

For more information about these templates, refer to the Mylyn documentation.

A dedicated view allows displaying the commit history. This window can be opened with the contextual menu called on the semantic model.



This view is particularly useful to monitor the current changes on the shared model. The objective of this history is also to attach as a change log when pushing back file-version of the model to Git.



This view is divided in two parts :

- The left part **list all the commits (saves)** that occurred on the Capella Project. Each commit is defined by the date of the commit, the user that committed the change (only if the Server supports authentication) and the first line of the Commit description associated to this commit.
- The right part describes **the impacted elements** by the selected commit(s) and the nature of their change (CREATED/DELETED/MODIFIED/UNTOUCHED).

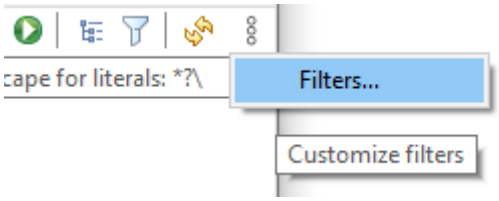
The Commit History View contains several buttons to modify the context of the commits list, filter those commits or modify the change viewer tree layout/content.

In particular, a "Filter" button is present in the Commit History view toolbar and allows the user to filter the content of the impacted elements.

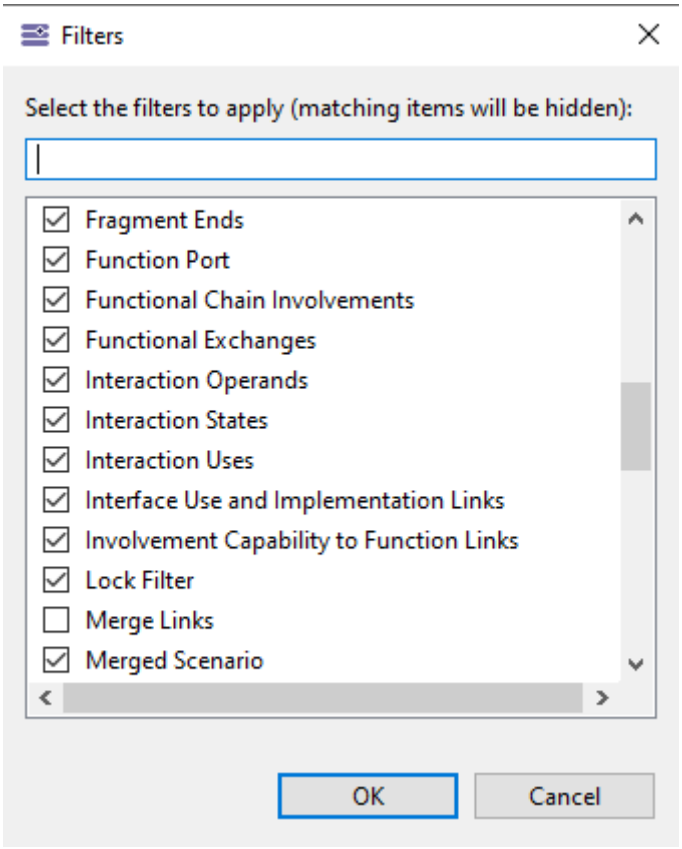
This button is represented by the following icon :

By activating or deactivating this button, the user can apply or not the selected filter.

Selected filters can be customized into the menu icon > Filters...



A new selection dialog is opened. From this dialog, the user can select filters to activate for the Commit History view. Filters provided in this selection dialog are the same as filters available in the Capella Project Explorer.



Session Details Properties Pages

The properties page (contextual action) on aird files of Capella connected project has a tab named *Collaborative Session Details*. It presents the repository information (location, port and name) and information about connected users and locked elements for this connected project. For more details, refer to Collaborative Session Details of the Sirius Collaborative Mode user documentation.

The properties page (contextual action) on aird files of local or connected Capella projects has a tab named *Sirius Session Details*. It provides a lot of useful information about the project (used viewpoints, information about representations and capella models). For more details, refer to Sirius Session detailed information of the Sirius user documentation.

3.6. Use Images in Remote Models

Images can be used

- in diagrams when the node is represented by an image
- using the Properties view, in rich text description of some elements such as Capella elements or diagram

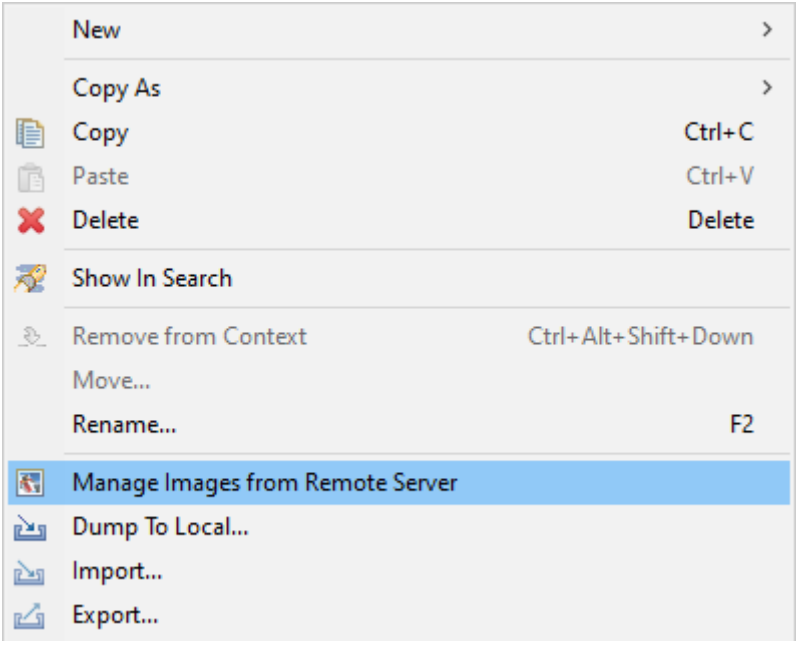
To use images in remote models, **only images that exist on the repository can be used**. Images from the workspace or from a local directory must be uploaded to the server in order to be used in a remote model.

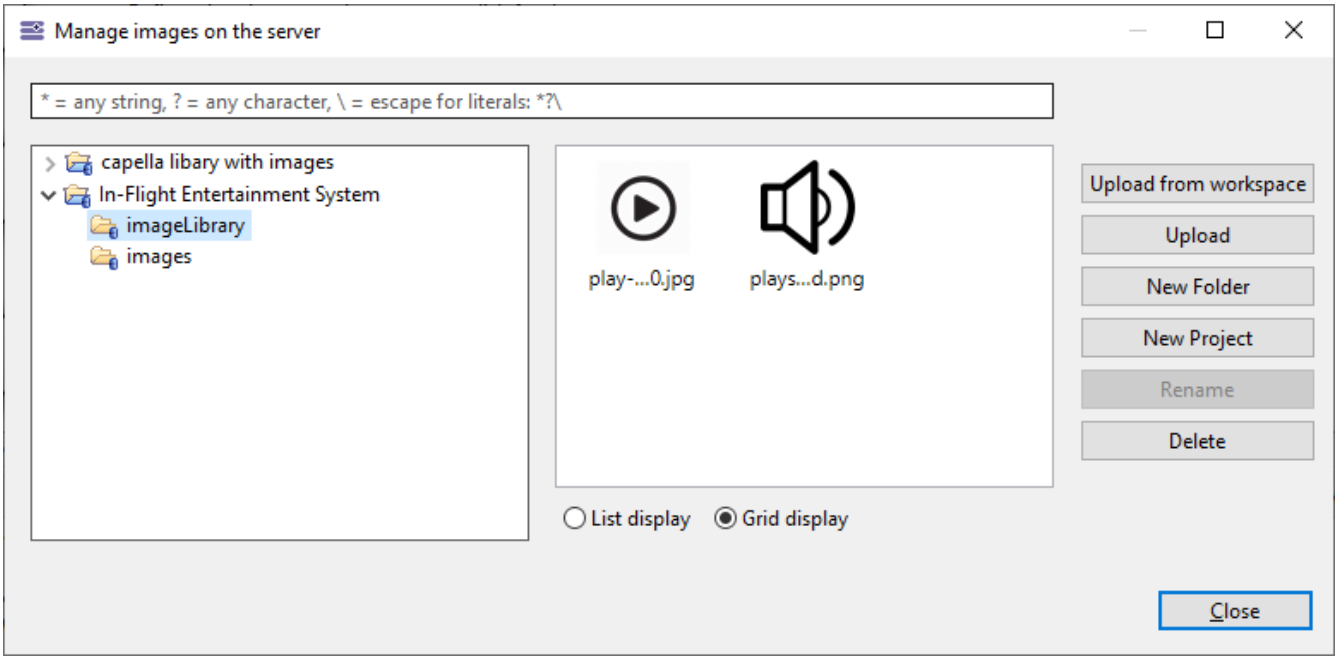
Manage images on remote repository

Manage images for an existing remote project

Once the project is exported, it is still possible to manage images on the server with the **Manage Images from Remote Server** dialog.

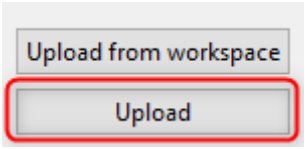
This dialog is available from the contextual menu on a shared aird file or an open connected project.



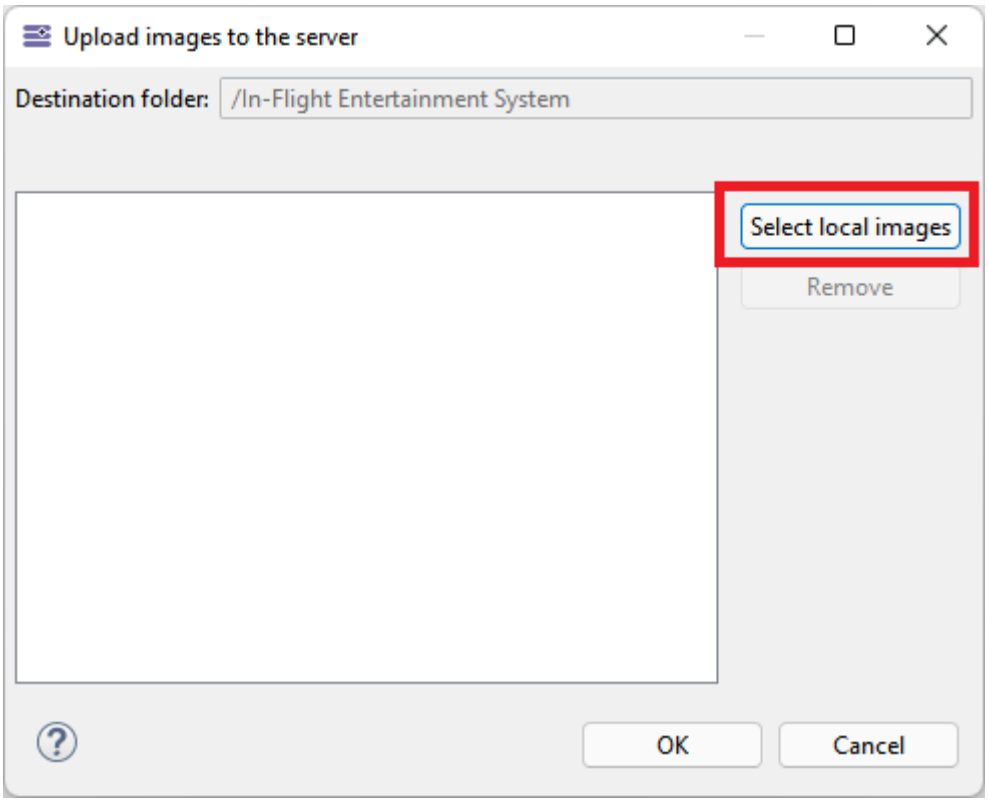


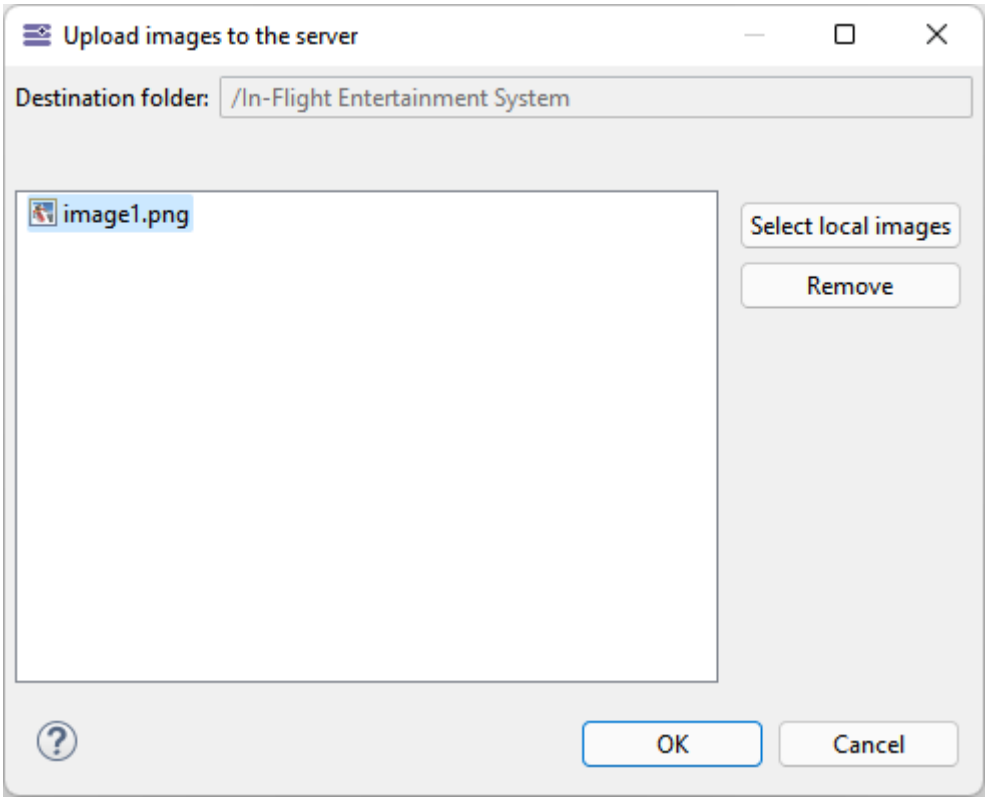
Uploading images from file system

- From the dialog, select the project or the folder where you want to upload the image and click on **"Upload"**. You can also create another project or folder with the "New Project" and "New Folder" buttons if you want to add images to another location:

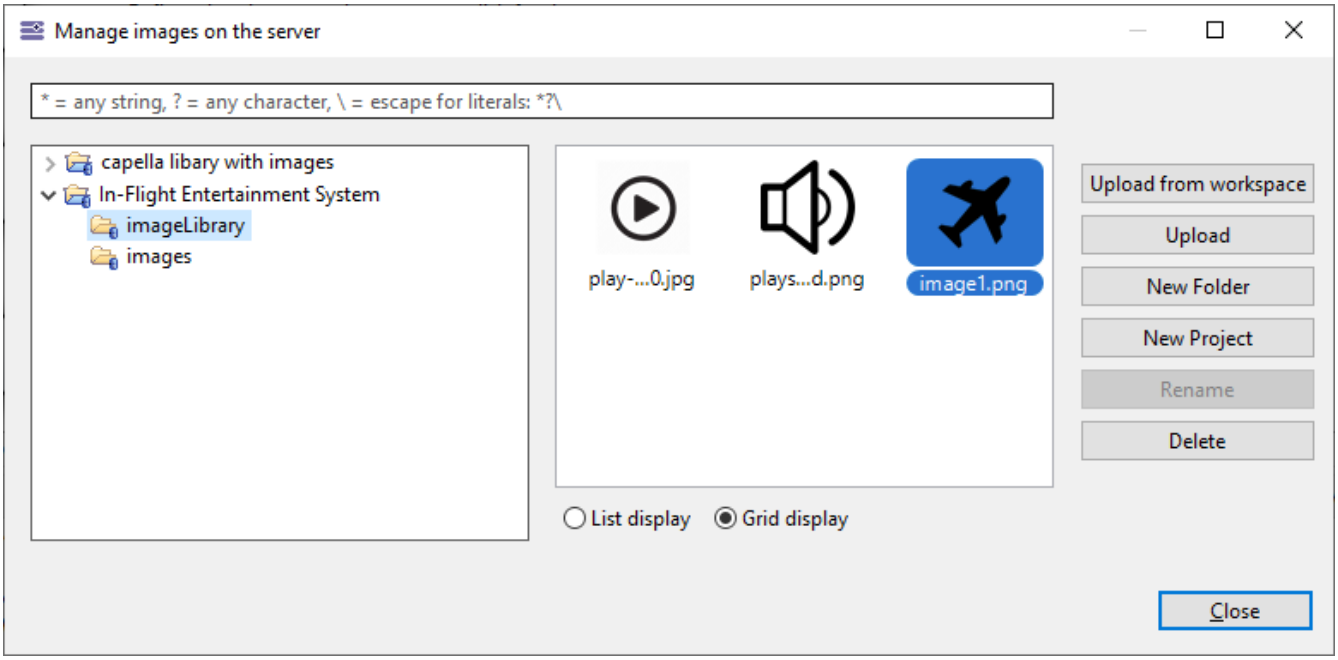


- Then click on "Select local images" in the "Upload images to the server" dialog to open a file system dialog explorer to navigate and select images you want to upload. Supported images format are **JPEG**, **JPG**, **PNG**, **SVG**. The maximum size of uploaded images is 10 MB per image. If greater, images are not displayed in the selection UI and cannot be exported to the server. This value can be changed by overriding the preference PREF_MAX_KILOBYTES_IMAGE_SIZE.





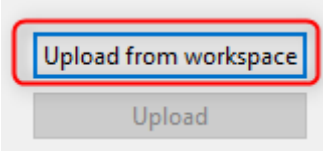
- Click OK and your image is uploaded on the server. Select the project or folder where your image is located and select it in the image gallery:



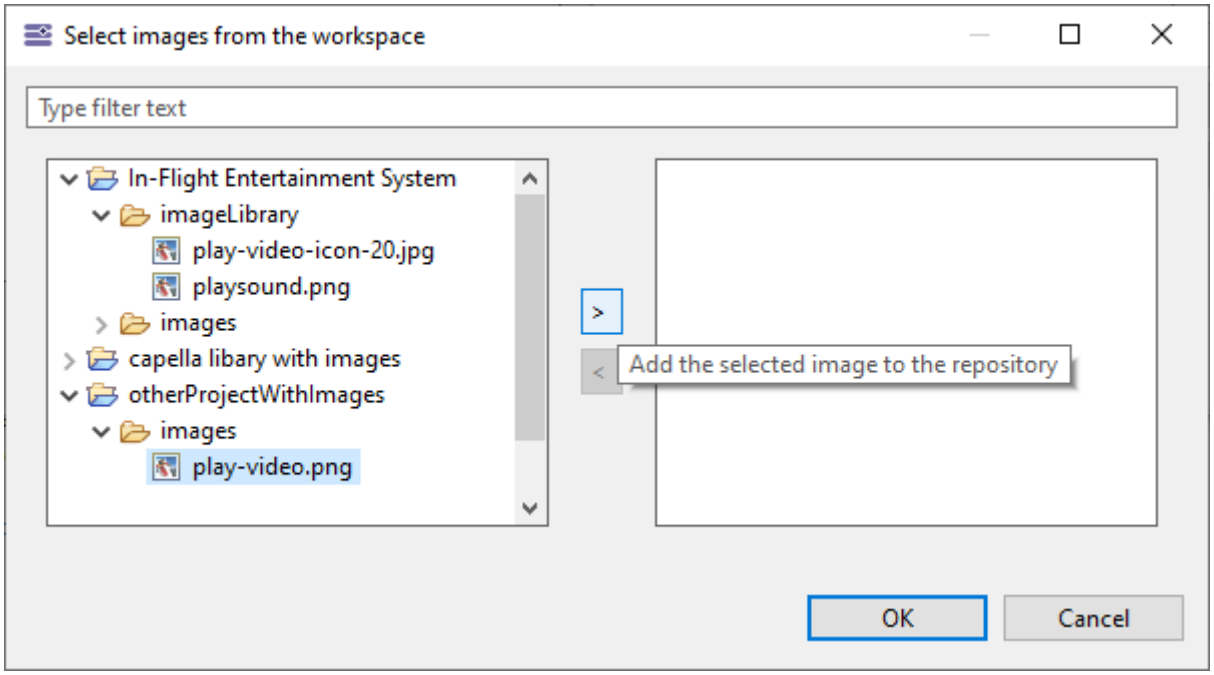
Uploading images from the workspace

It is also possible to upload whole sets of images by selecting a project, folders or single images from the workspace

- From the dialog, click on **"Upload from workspace"**



- Then use the button to add an image, a folder or a project



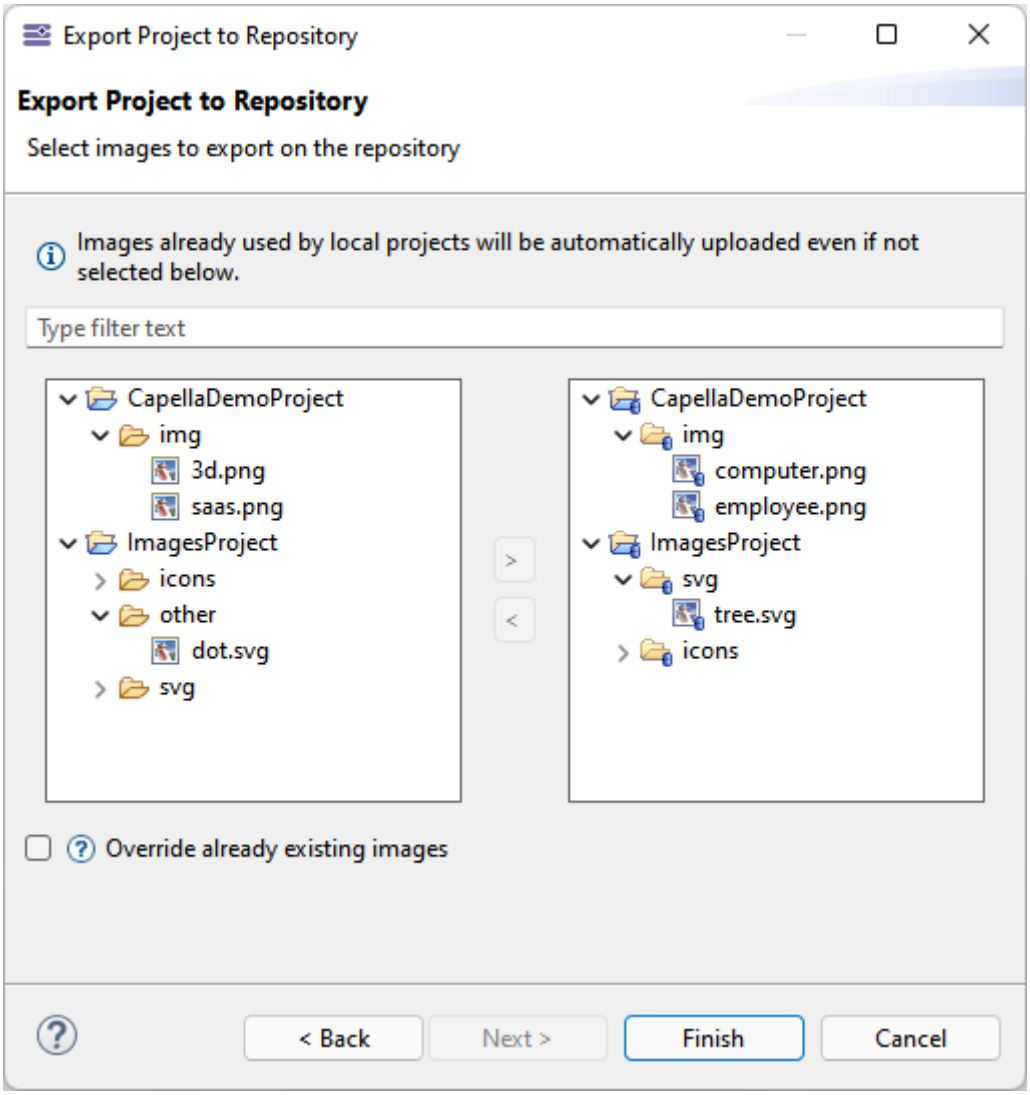
The image hierarchy of uploaded images(project and folders) is identical to the selection in the workspace.

How to Change an Image Already on the Server

An existing image can be overridden on the server. All the diagram elements, in the shared diagram, using the replaced image, will be automatically updated.

Export images to the server when exporting the project

Export images wizard page



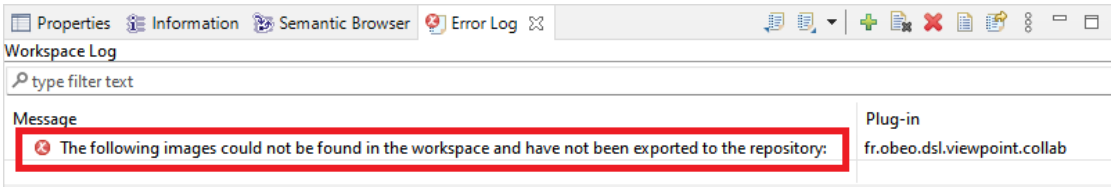
On the Export project wizard, you will be able to choose the images you want to export to the repository in this new wizard page.

The images used by the exported projects will be automatically exported to the repository to keep the consistency of the shared representations. This means that if you explicitly use an image in one of your projects to export, this image will be exported even if you didn't select it.

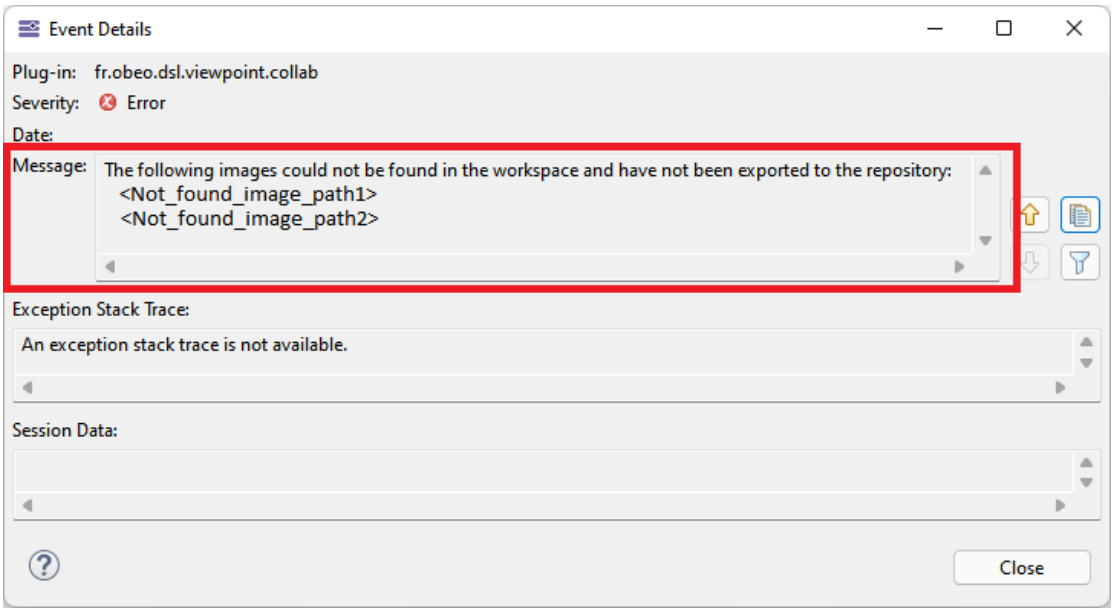
The left panel shows the existing images in the open workspace projects, and the right panel shows the images you have chosen to export from the left panel. The "**Override already existing images**" checkbox allows you to override existing images on repository that have the same path as those added to the right panel.

Images in **JPEG**, **JPG**, **PNG** and **SVG** format are supported. The maximum size of uploaded images through the export wizard is 10 MB per image. If greater, images are not displayed in the selection UI and cannot be exported to the server. This value can be changed by overriding the preference PREF_MAX_KILOBYTES_IMAGE_SIZE.

If the referenced images do not exist when exporting the project to the server, an error appears in the "Error Log" listing all missing images.



Open the error details to see all affected images:



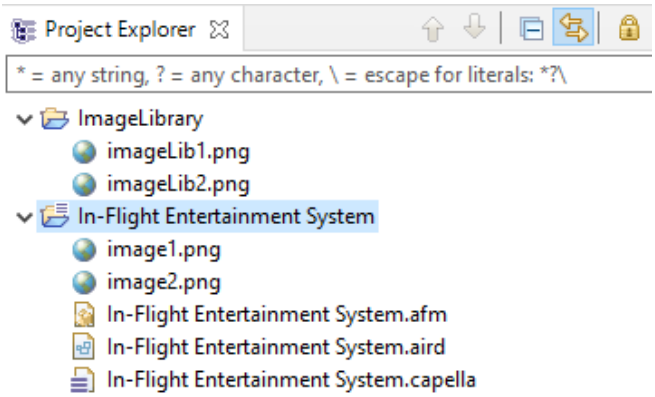
If an image that has been exported to the server is afterward not used anymore in a remote diagram, then this image will not be imported when importing the project if you choose the **Import only used images** option in the import wizard.

Images used before exporting the project to the server

When a model is exported to the Team for Capella Server, referenced images which are available in the workspace will be exported along with the model. In the local project, it is important to select images in the right project because it will drive the way the image is recreated when importing the project locally (after it has been exported to the server).

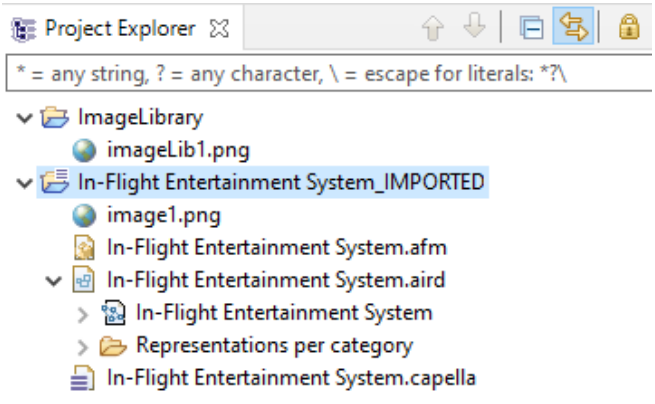
- If the image has been **selected in the current local project** about to be exported, then once imported, the image **will be located in the imported project**.
- If the image has been **selected in another local project**, then once imported, the image **will be located in the same other local project**.

Local project where images, image1 and imageLib1, have been used as workspaceImage before exporting:



Projects after exporting then importing the remote project:

Note that only used images have been exported then imported



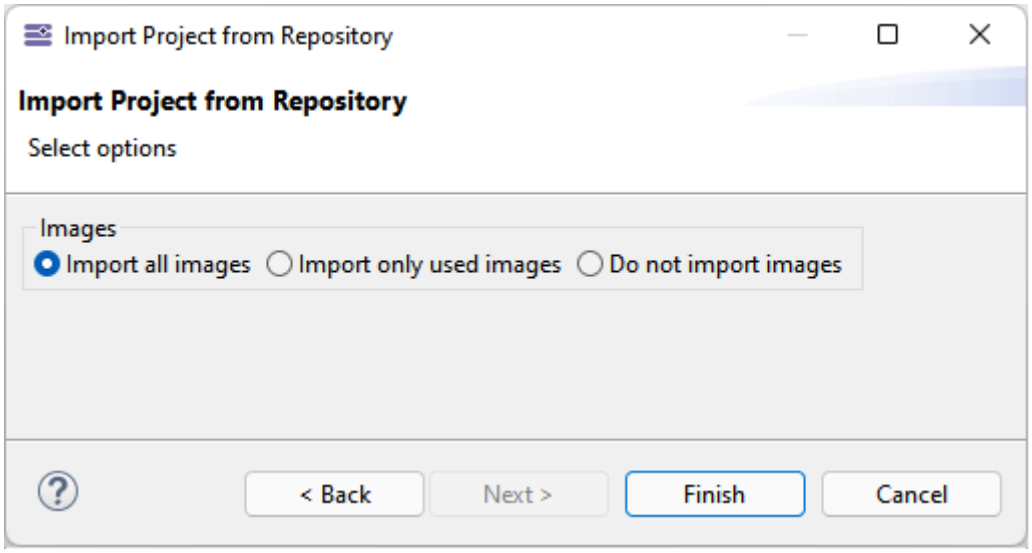
Import images from the server when importing the project

Importing images is done when importing a remote project in the workspace using the Team for Capella import wizard.

When importing the remote project locally, the imported images will be created in local projects that correspond to their location on the server.

The import wizard allows you to choose from three different options for importing images:

- **Import all images**: import all images existing in the repository.
- **Import only used images**: import only images used by the project and its dependencies.
- **Do not import images**: Do not import any image.

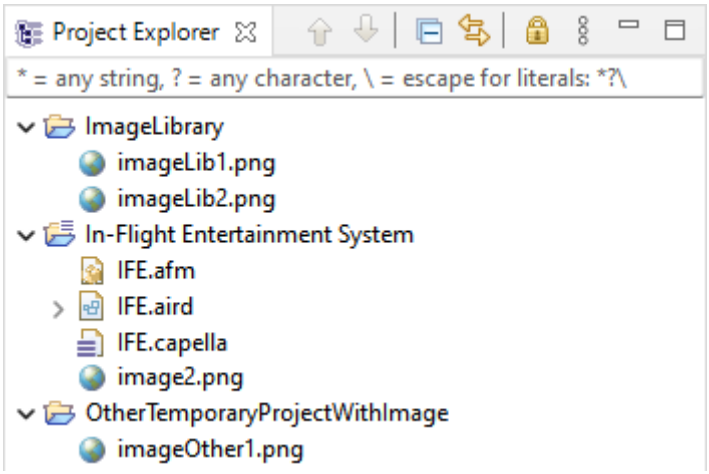


Images that already exist on the workspace will be overridden automatically.

Import images options

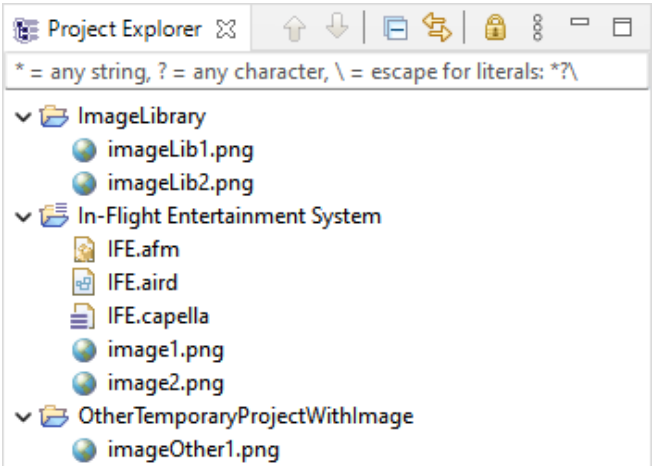
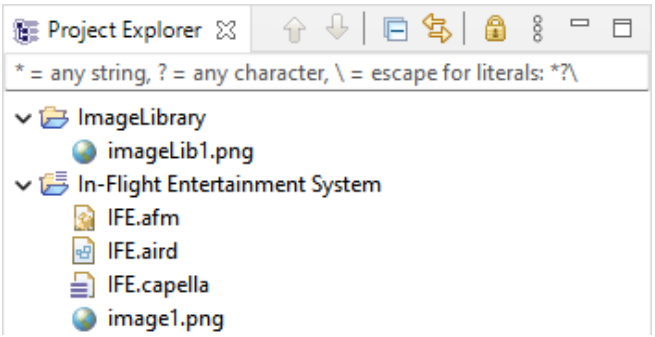
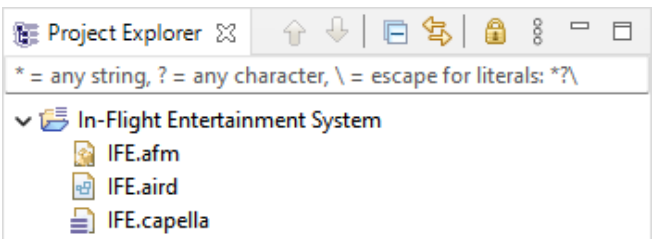
Starting from a local project, all images in the workspace have been exported to the server with the project.

Suppose that **/ImageLibrary/imageLib1.png** is referenced by the project, and **/In-Flight Entertainment System/image1.png** has been exported because explicitly chosen in the export wizard page.



Let's consider that the local workspace is then completely cleaned up to import the remote projects.

The result of the import will be different, according to the selected option:

Import all images	
Import only used images	
Do not import images	

- When importing the project locally, it will also create projects containing the referenced images. These projects are also zipped by the importer job. See archiveProject parameter in Importer Parameters chapter.
- By default, the importer job uses the **Import all images** option, this option is not yet configurable with a specific parameter.

Images on the Team for Capella Server: What to retain in few words

What to retain in a few words:

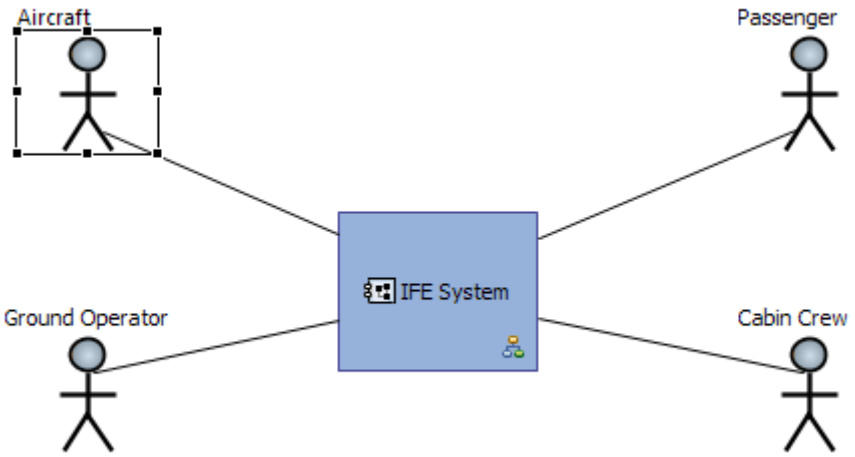
- Only images that exist in the repository can be used.
- To upload images to the server, they must be selected manually when exporting a project from the **Select images to export on the repository** page.
- It is also possible to manage images on the server from the **Manage Images from Remote Server** context menu, available from a shared aird file or an open connected project.

Images used in diagrams

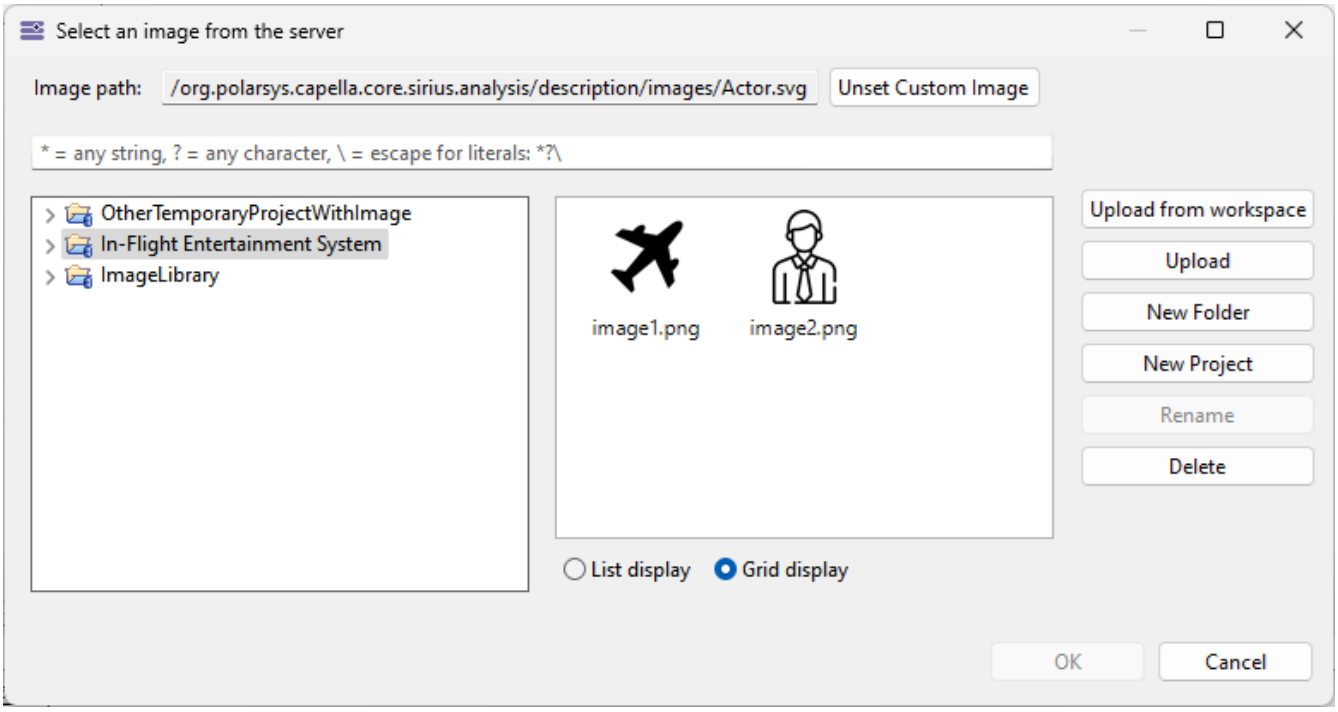
To use images in remote models, only images that exist in the repository can be used. Images from the workspace or from a local directory must be uploaded to the server in order to be used in a remote model.

In a diagram it is possible to associate an image to a node using **"Set style to workspace image"**

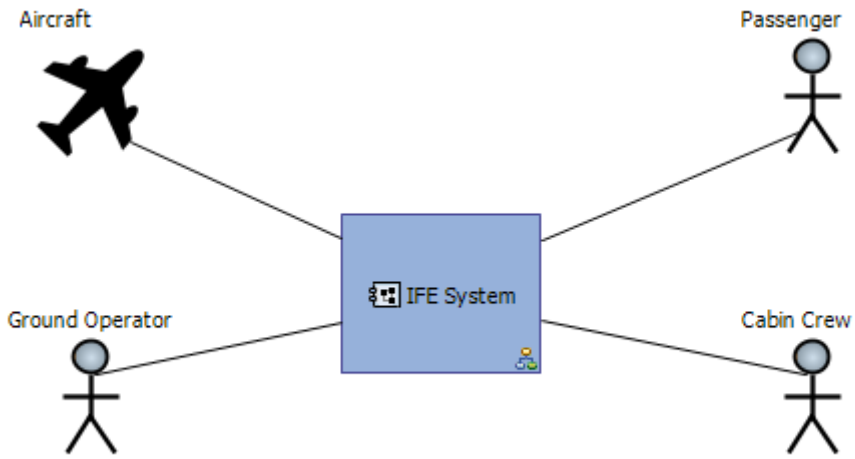
- Open a diagram in a remote model and select the element on which you want to put the image:



- Then click on the "Set style to workspace image" button from the diagram toolbar which will open the "Select an image from the server" dialog. Select the project or folder where your image is located and select it in the image gallery:



- Click OK and the image is then updated in the diagram:



From this dialog it is also possible to manage remote images. Refer to Manage images on remote repository documentation

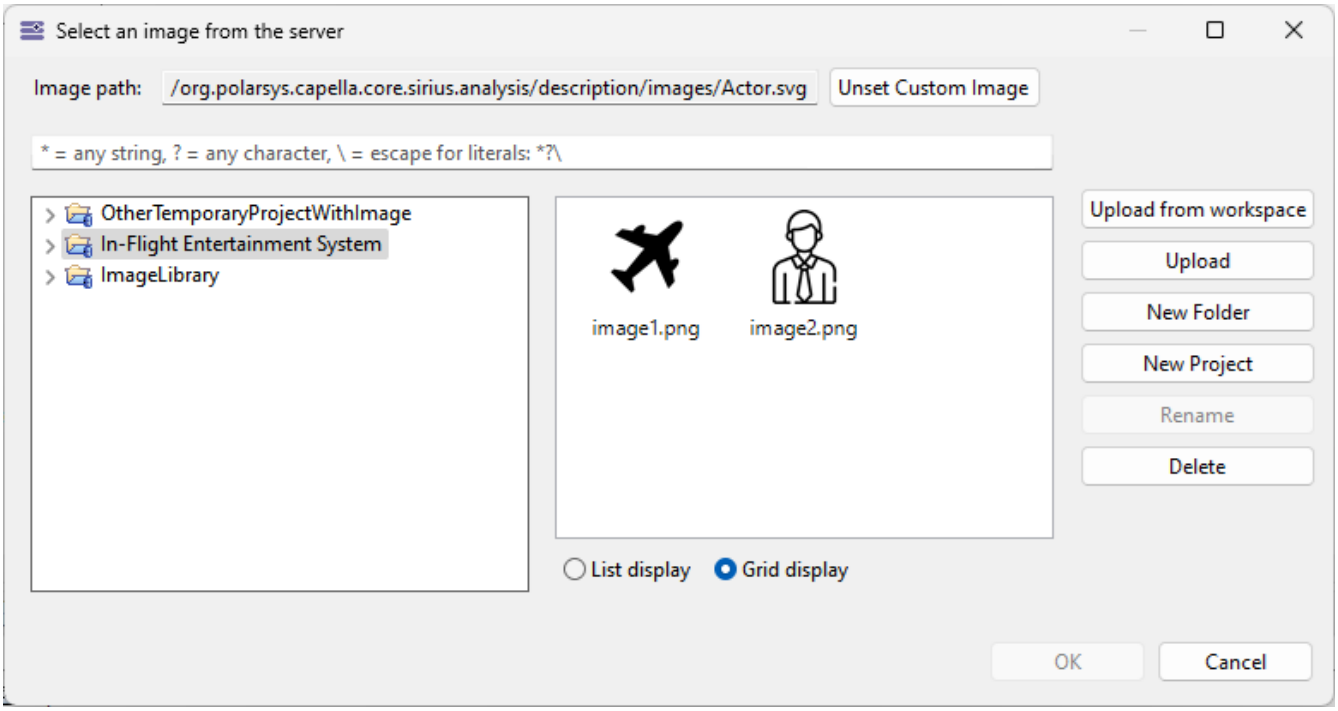
Images used in Capella description editor

It is possible to add a description with images, for any element of a Capella project, using the **description** tab in the Properties view.

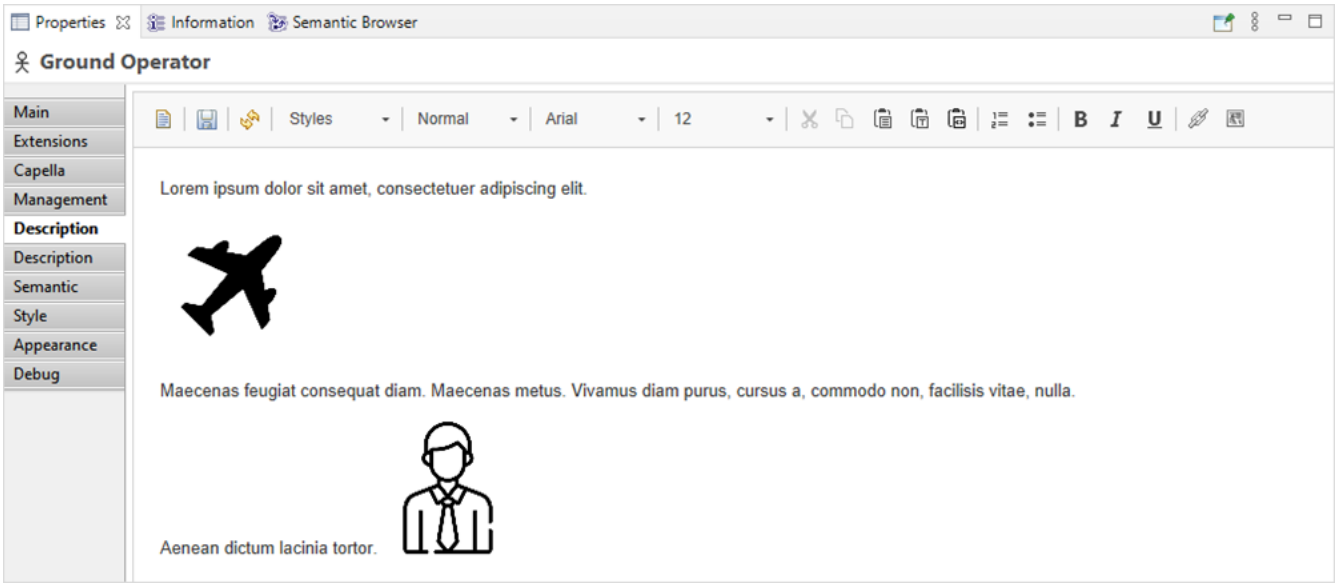
Like in remote models, only images that exist on the repository can be used. There are two ways to add an image in the description:

- Add an image using the "Select an image from the server" dialog
- Copy and paste an image that will be automatically exported to the server in the **<connected project name>/images** folder. This image will be available for selection as any other images on the remote repository.

To add an image with the selection dialog, click on **Add image** button  and choose the image.



Images are then added to the description:



3.7. Working with Libraries in a Multi-user Context

Export Procedure

One classical pitfall is to export models (libraries and projects) that are linked by "reference" relationship one by one. Rather, export of linked models must be done at the same time because doing it one by one may lead to the export of still exported models. For the sake of illustration, having two projects P1 and P2 referencing library L1 may lead to one re-export of L1 if one tries to export P2 after having exported P1. The following section describes the correct procedure.

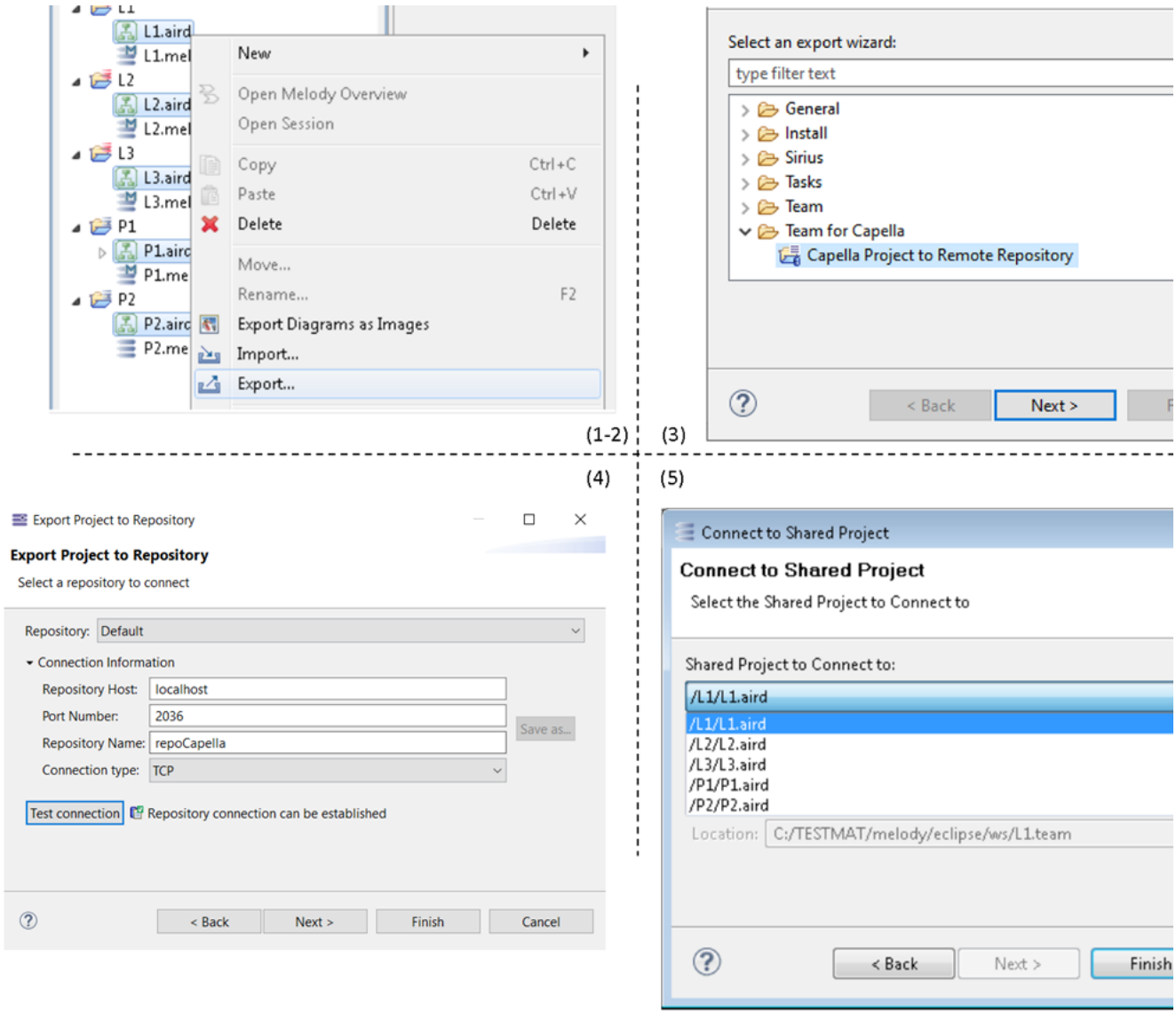
We assume in this section that a Team for Capella Client is opened and its workspace contains a set of models (projects and libraries) that are interconnected by the way of reference links.

In that context, the export procedure is as follows:

1. Select all AIRD of interlinked models,
2. Right-click on the selection, and click on export,
3. Choose Export model in Team for Capella category,
4. Test the connection, authenticates if required and click on finish.
5. You can afterwards connect to the models you want as usual.

Figure below illustrates the four steps described above in the given context:

- five models (two projects P1, P2 and three libraries L1, L2, L3),
- P1 refers to L1 and L2,
- P2 refers to L1,
- L2 refers to L3.

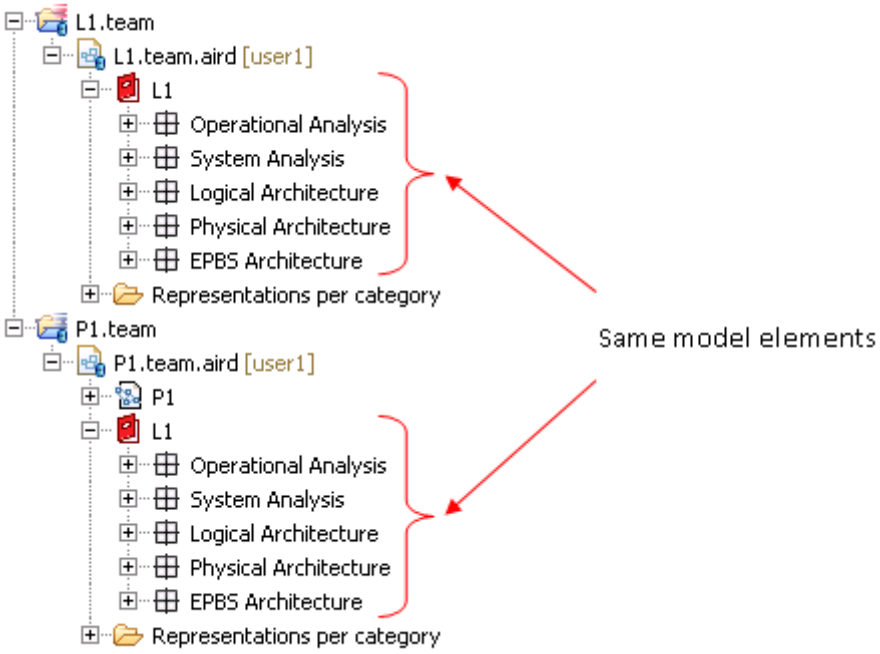


Project/Library Usage

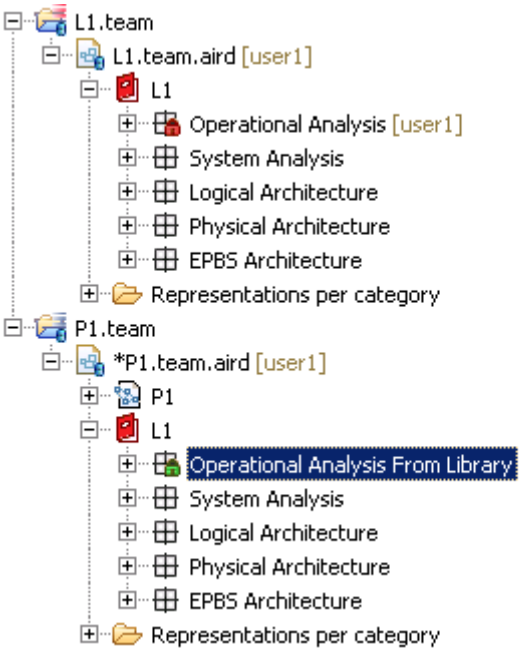
Libraries can be accessed as classic remote projects with Team for Capella and have almost the same behavior as with Capella standalone:

- When connected to a remote library, its semantic model is visible as well as all its diagrams,
- When connected to a remote project referencing libraries, only semantic models of these libraries are visible from the project.

It is allowed to open, in the same client, a project and some libraries it references. Thus, it is possible to have two views (or more) of the same semantic elements:



If a library is referenced with a "readAndWrite" access policy, it is allowed to change its semantic model from the project connection, from P1.team in this example:



Even if the user is logged with the same login to L1 and to P1, if a change is done on one side, there will be a green lock on this side and a red lock on the other (so concurrent changes are forbidden on library's elements).

Limitations and Known Issues

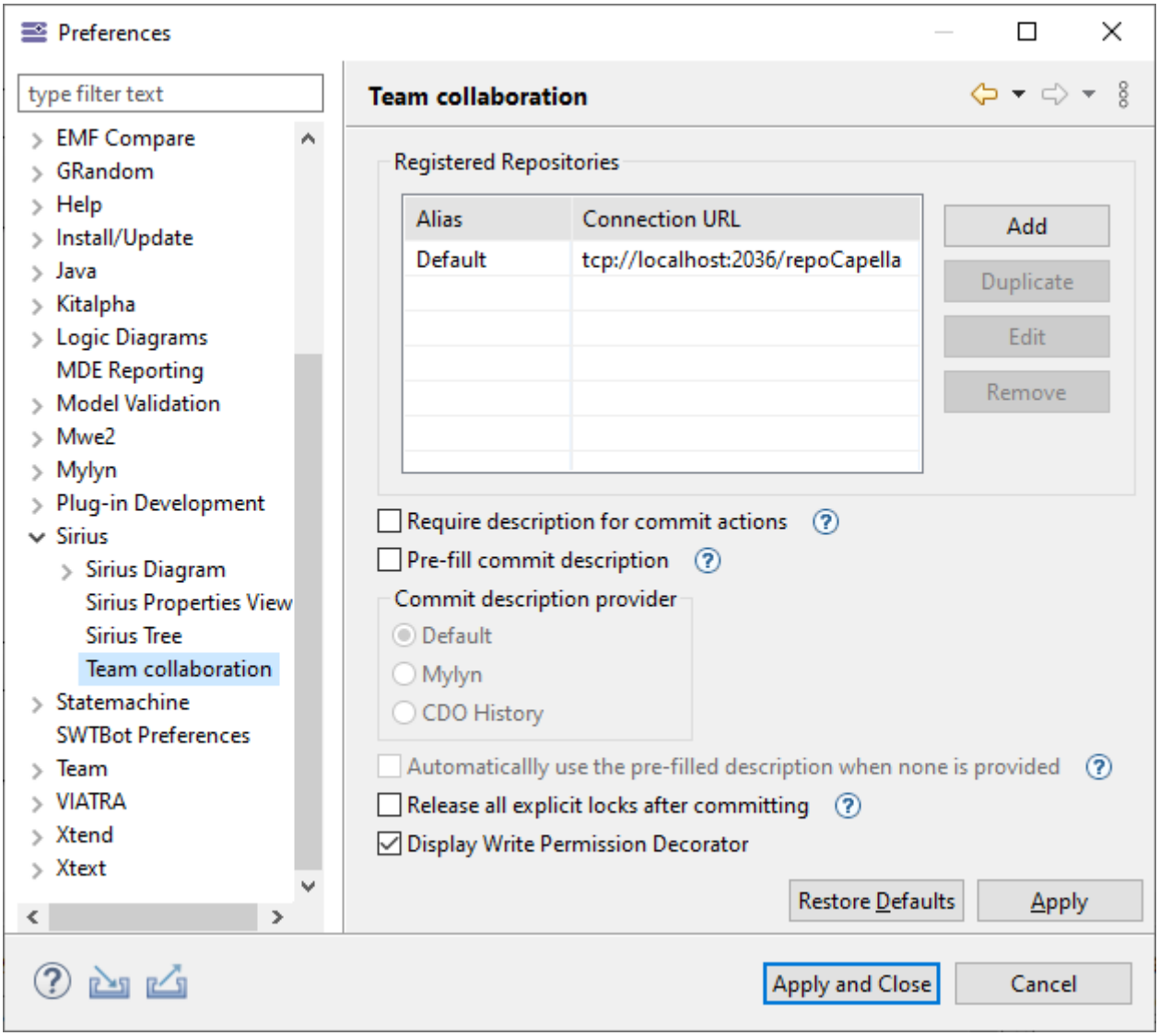
- When working with a library, it is recommended to close referencing projects (this recommendation applies only when several remote libraries/projects are open).
  - Known issue: lock decorators are not correctly updated between a library and its view in referencing projects,

3.8. Client Configuration

Preferences

Team Preferences

Team Preferences are available in Window / Preferences / Sirius, section **Team Collaboration**.



The **Registered Repositories** section contains all saved server information. There is a default saved

repository that can be overridden only in this preference page. Registered repositories can be edited, duplicated or removed, and new repository configurations can be added. All these configurations can be retrieved in the Connection / Import / Export wizards.

The checkbox "**Require description for commit actions**" specifies whether a dialog allowing to input a description when committing should be displayed systematically or not.

By activating the preference "**Pre-fill commit description**", any time the user is asked to enter a commit description, the framework will compute one using a list of registered participants (see description below). This description will be presented to the user, so he can modify it or simply reuse it for its current commit.

- **Default**: This entry is a strategy used to select the most suited participant. It selects the first participant that can provide a commit description for the current context. It iterates on all registered participants until one can be activated (Mylyn, CDO History etc...). It starts from the one registered with the lowest priority in the extension point. The order of priority is represented by the order in the list below the "Default' entry in the preference page (first at the top).
- **Mylyn**:This entry uses the current activated Mylyn task to build a commit description. It only uses tasks that are not completed. If there is no active (not complete task), it provides an empty description. The description can be customized using the template defined in the preference page Mylyn > Team. *Activation criteria:* There is an active Mylyn task
- **CDO History**: This entry uses the CDO History of the current repository. It gets the last commit description entered by the current user and uses it as pre-filled commit description. It is only activated if the current session uses authentication. This participant also excludes commits that are tagged as technical commits. *Activation criteria:* The user is authenticated on the CDO Server.

By activating the preference "**Automatically use the pre-filled description when none is provided**", any time the user commits and do not specifically provide a commit description, the description computed from the mechanism described above will be used.

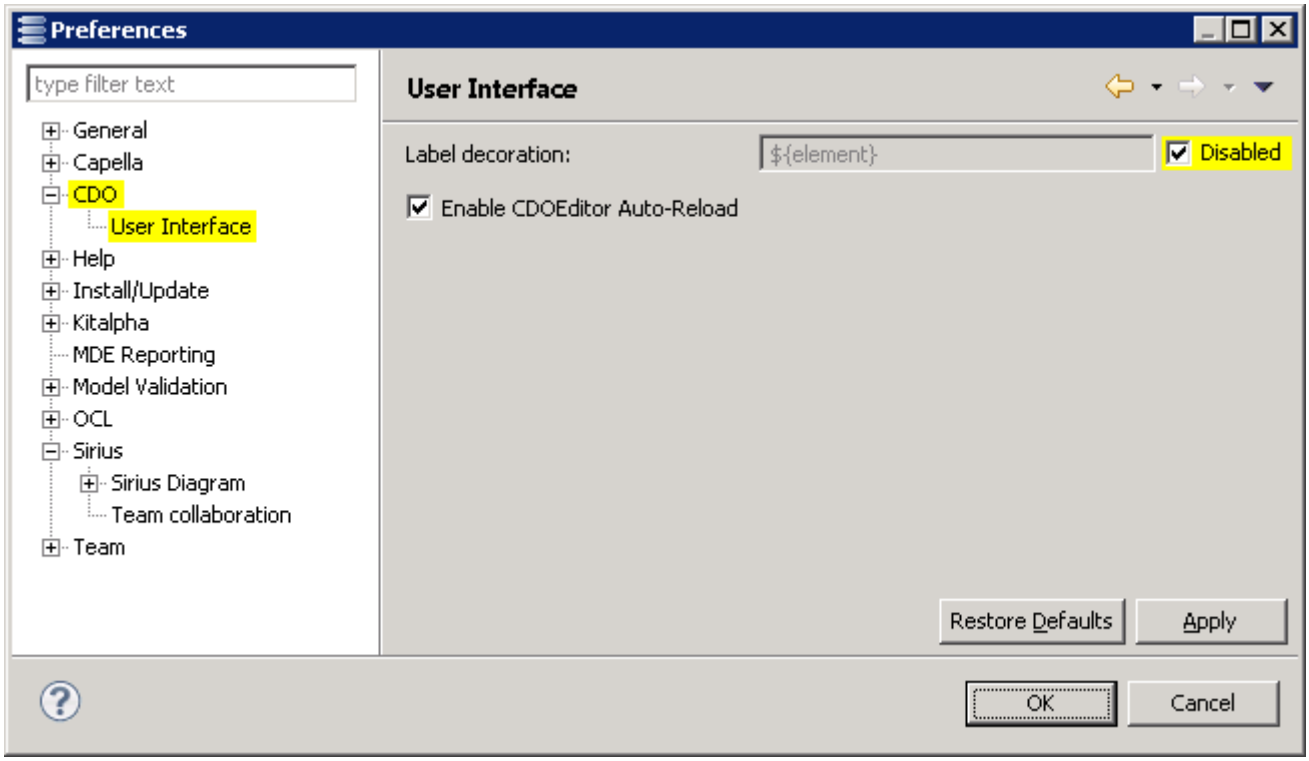
The system property `fr.obeo.dsl.viewpoint.collab.common.commit.description.max.length` can modify the length of the commit message. By default the value is the Integer maximum value. **This property needs to be set to the same value on the client and the server**.

This default Integer maximum value length is due to com.thalesgroup.mde.melody.db.h2.H2Adapter that consider the commit comment as a CLOB. Note that this is a custom H2Adapter for Team for Capella. It replaces the default org.eclipse.net4j.db.h2.H2Adapter that expects a VARCHAR for the comment description DB field limiting the length of the message to 255 characters. If the commit description is longer than the accepted max length, it will be truncated before commit to match the max length in core wizards, actions and session save operations. For components extending the collaborative layer, if they directly call setCommitComment() and commit() methods on the CDO transaction, they can use

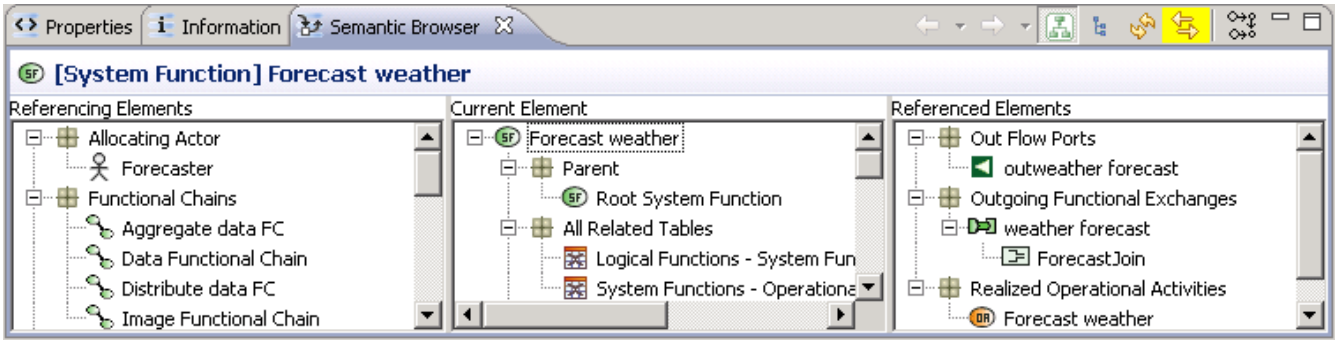
`fr.obeo.dsl.viewpoint.collab.common.internal.commit.CommitCommentUtil.fitCommitDescriptionLength(String)` to fit their commit comment. -1 can be used to remove the limitation. Otherwise only values greater than or equals to 10 are accepted. "Xxxxxxxxxx" will become "Xxxx [...]". If the property value if -1, it will take the system dependent SWT widget Text.LIMIT length.

Other Preferences

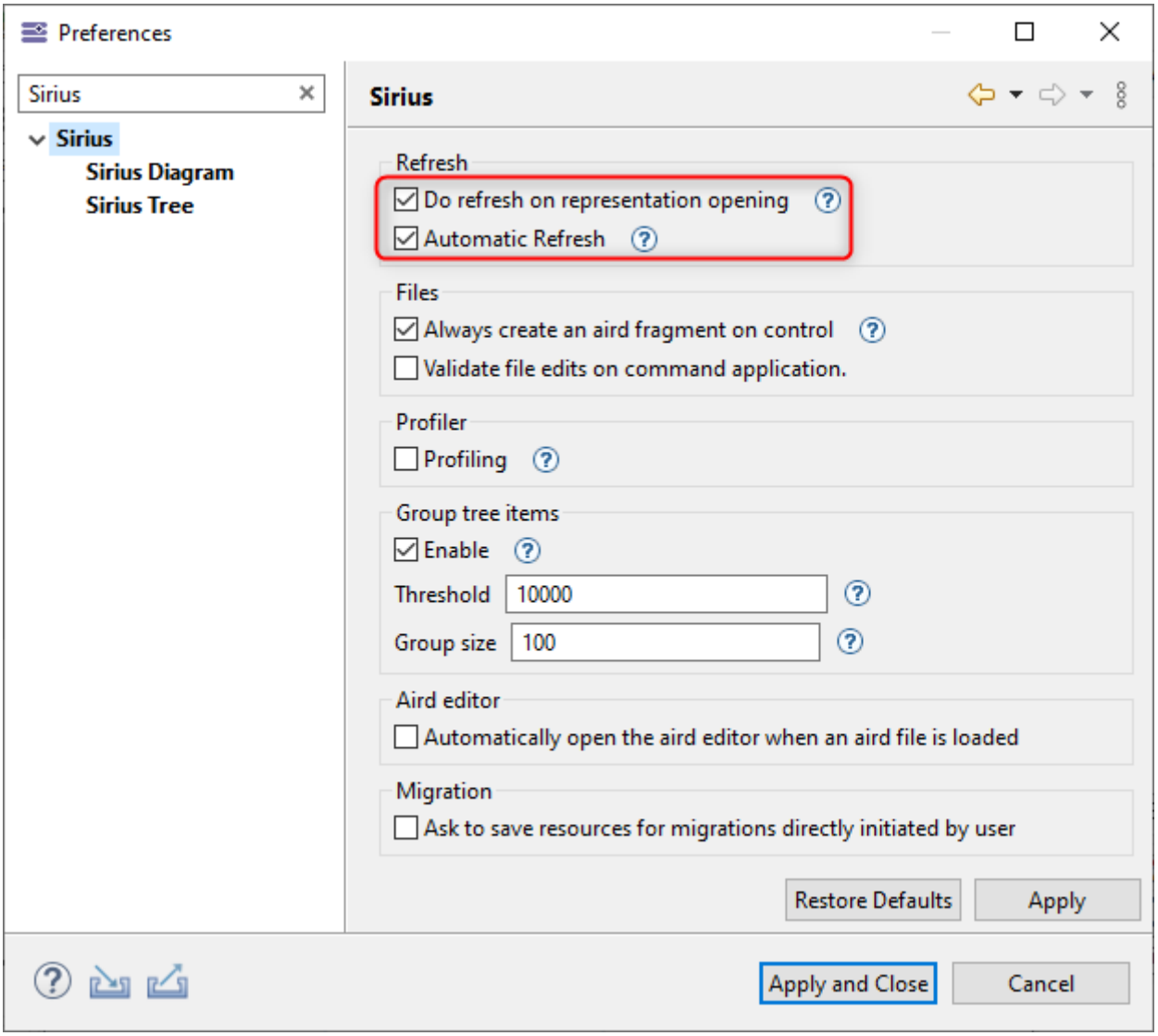
Please check the following settings in the other sections of the Preferences.



For a better reactiveness of the whole workbench, the synchronization of the Semantic Browser should be disabled. Reminder: when the Semantic Browser is not permanently synchronized, typing F9 focuses the Semantic Browser on the currently selected element.

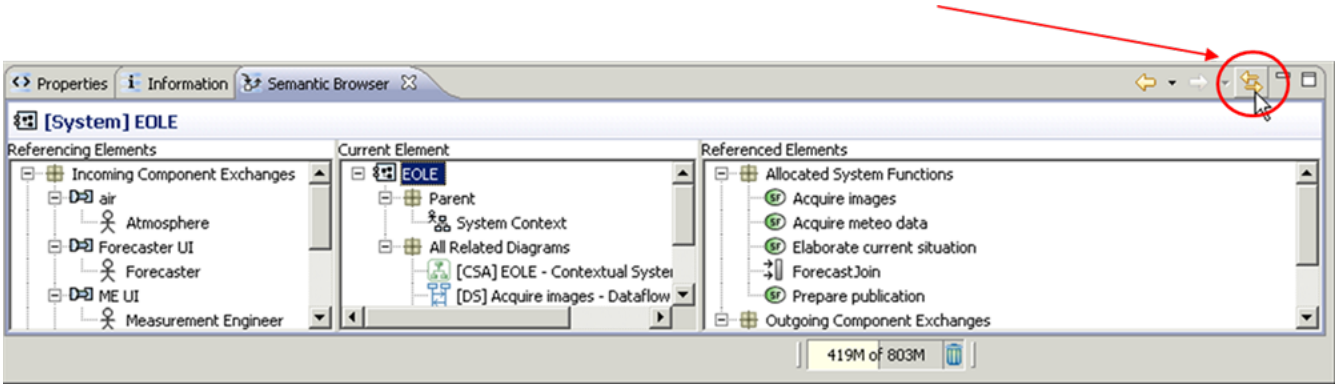


"Automatic refresh" and "Do refresh on representation opening" are activated by default as it is in Capella.



They can nevertheless be overridden at the project level.

Automatic synchronization of Semantic Browser is deactivated by default.



Configuration Project

A Capella Configuration Project cannot be shared through several users by exporting it to the Server.

To use the Capella Configurability feature in Team for Capella, the Capella Configuration Project needs to be referenced on each Team for Capella connection project.

VM Arguments

The client behavior can also be set using VM arguments added to the capella.ini or in a launch config.

By checking "Remember me", you can store your username and password in the Eclipse's Secure Storage. If you do so, your username and password will not be asked for future connections.

3.9. Change management

Introduction

Change management is about adding extra information about users activities while modeling. They can be related to any aspect of the modeling session (current tasks, current teams, a more detail explanation etc...). Its integration in Team for Capella provides a way to:

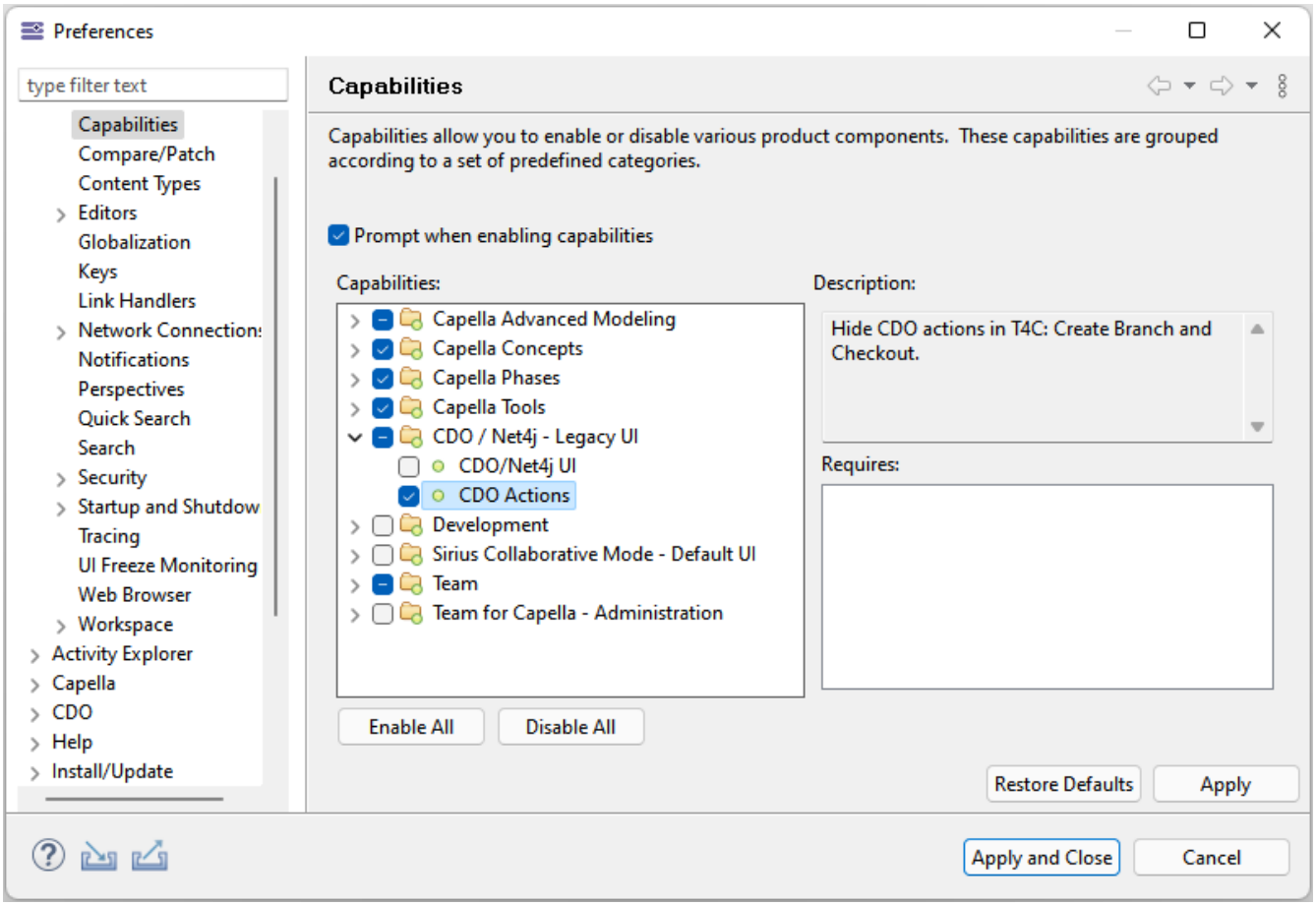
- Ease the way users fill them.
- Structure and request them.

Those information are attached to a commit. They can be visualized in the *Commit History View* by selecting each commit. Be aware that some commits are made by modeler itself. They do not represent commits that users would have made. They are tagged with the property *team-technical-commit : true*.

Main documentation

The main documentation of the *Commit History View* is available in the corresponding section of the Sirius Collaborative Mode user documentation.

Note that some actions have been hidden in Team for Capella, such as *Create Branch...* and *Checkout* popup menus. You can enable the **CDO Actions** capability in the Preferences page to access them.



Filling up extra information

In Team For Capella, there are two ways to fill up the extra information attached to a commit.

- Users can choose to enter it for each commit using the preference "Require description for commit actions". The process can be eased using "a Commit description provider".
- Users can choose to automatically use the description computed by the framework using the preference "Automatically use the pre-filled description when none is provided". Then use the custom action "Save with Description" when they want to change or add more detail about the current activity.

The following sections explain the different facilities used to compute a commit description.

Using CDO History

This strategy uses the history of the Team for Capella Server to guess what information the user wants to enter. Before each commit, it will look for the last commit done by the current user (that is not a Technical commit). For example, let's say the current user is *user1* and the server has the following history:

Date	User	Description
31/08/2017 16:00	User1	Update Xmi Ids team-technical-commit : true

Date	User	Description
31/08/2017 15:59	User2	Activity 2 Doing some work
31/08/2017 16:58	User1	Activity 1 Doing some other work
31/08/2017 16:57	User1	Activity 1 Doing some other work

If *user1* saves the model, the framework would compute the following commit description:

```
Activity 1
Doing some other work
```

If he has activated the preference "Require description for commit actions" a dialog will open suggesting this message.

If not activated and the preference "Automatically use the pre-filled description when none is provided" is activated, the commit will be made using this message as commit description.

To activate this strategy, go to the preference page: *Sirius* > *Team collaboration*. Select *Pre-fill commit description* and select *CDO History*. Be aware that this mode only works on an authenticated Team for Capella Server.

Using Mylyn

This strategy uses Mylyn tasks to compute a commit description. Using the template defined in "*Preference* > *Mylyn* > *Team*", it computes a commit description from an active and not completed task. This strategy is really handy when using "Automatically use the pre-filled description when none is provided" preference. Indeed, with this configuration, the user only has to activate or deactivate Mylyn tasks to have a clean history filled up with extra information.

To activate this strategy, go to the preference page: *Sirius* > *Team collaboration*. Select *Pre-fill commit description* and select *Mylyn*.

Export user activities

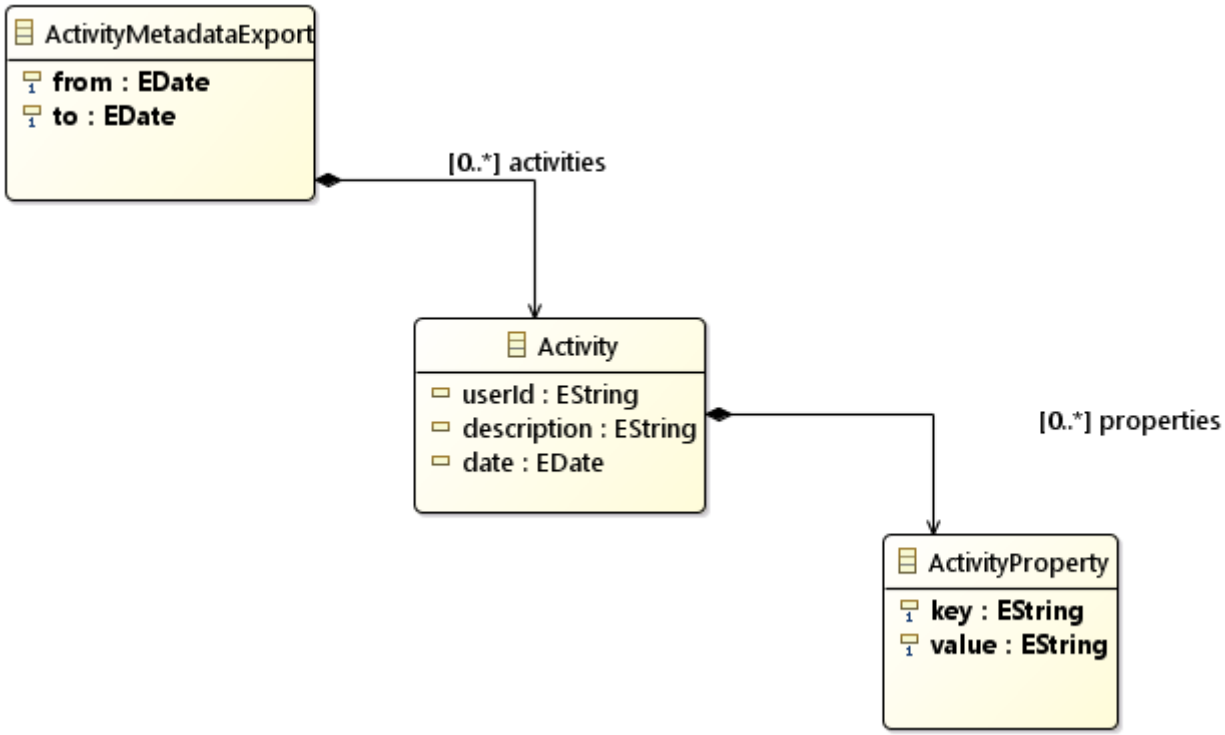
Once history filled up with meaningful information, the user might want to use it. To do so, he can export it to a model format using the "Export Metadata actions" from the *Commit History* view.

Another way to export metadata is by using the importer.

Use exported activities

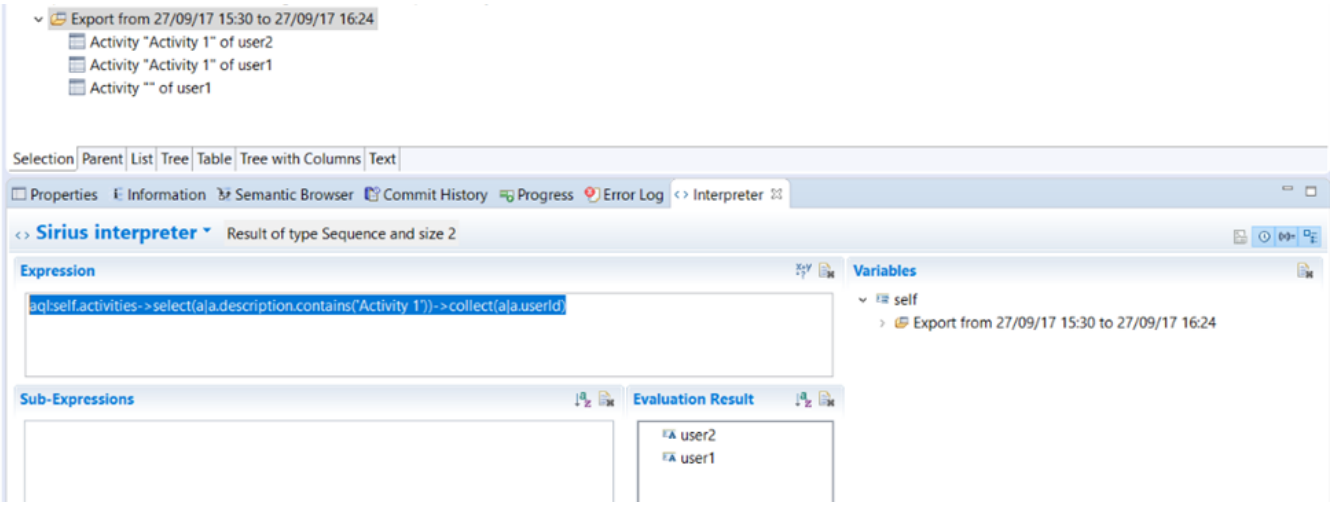
Once the information has been exported to a file, a model editor can be used to browse the

different activities that occurred on the server. Using the "*text*" tab, he has access to a textual representation of the current model. He can even request it using Aql requests (more documentation here). Here is a representation of the metamodel:



For example, he might want to request all users that have participated to a given activity. To do so, he could use the following AQL request:

```
aql:self.activities->select(a|a.description.contains('Activity 1'))-
>collect(a|a.userId)
```

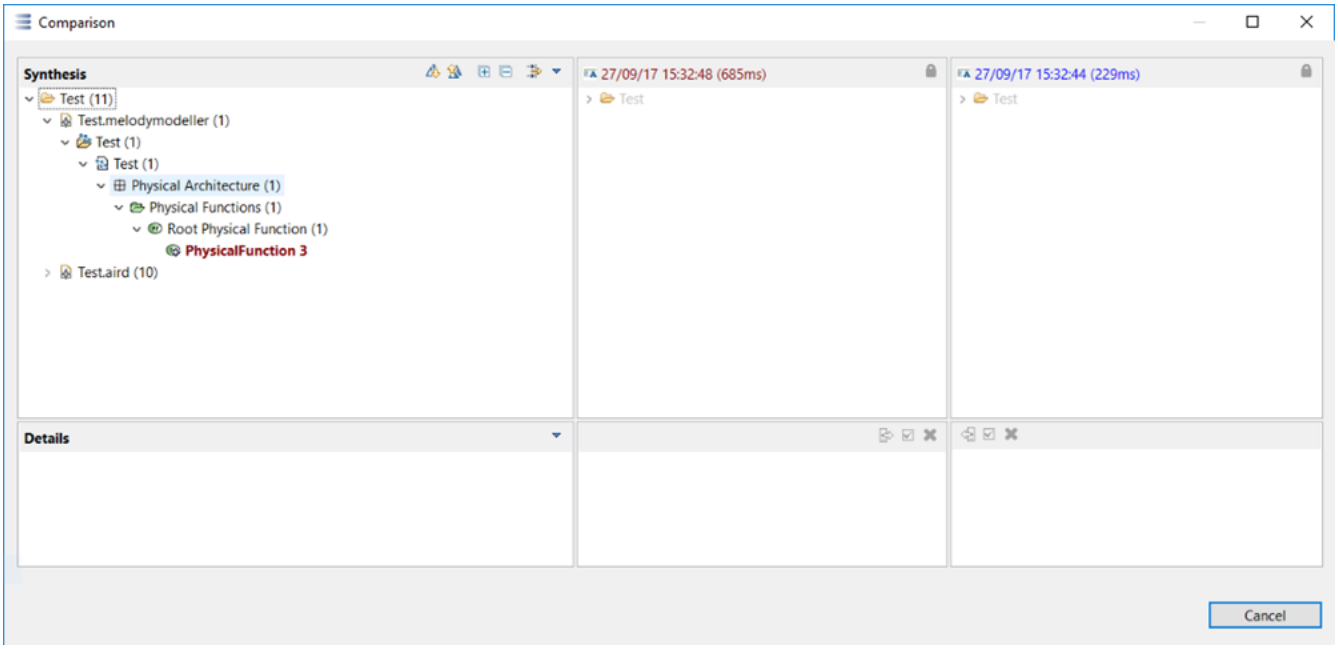


Using a dedicated format in the commit description (defined here), the user can even creates its own custom properties. Each one of them will be transformed into *ActivityProperty*. It might be used to create more advanced Aql requests.

Comparing commits

When using a server configured with Audit mode, it is possible to compare commits between each other. To do so, the user should open the Commit History view. From there he can select one or two commits and use "Compare with each other" or "Compare with previous" menus. The comparison is done using Diff/Merge framework (see document here).

Limitation: The Commit History View allows merging consecutive commits with the same user and description in only one visible commit. The Diff/Merge actions are not enabled on this kind of commit. You have to deactivate first the "Merge Consecutive Commits" option to make those actions enabled.



In the picture above, the differences are stored under two roots each representing a resource.

- Semantic resource (Test.capella): Under this root, all semantics differences are displayed.
- Graphical resources (Test.aird): Under this root all graphical differences are displayed.

Be aware that at this time, the integration between Team for Capella and Diff/Merge do not offer merge functionalities.

Chapter 4. Project Administrator Guide

Contents

4.1. Project Administrator Overview

Team for Capella installation can be completed with Jenkins used as a scheduler for various job managing the Capella project shared on a CDO server. Indeed, Project Administrators will find functionalities concerning:

- Server lifecycle management
  - The status of the server is visible with Jenkins, and there are jobs to start or stop it.
- Backups
  - Several jobs are available in Jenkins to back up the shared Capella project, to back up the SQL database created from the shared Capella project, to back up the model defining users and roles.

4.2. Jenkins Configuration

Team for Capella Scheduler

Team for Capella provides many applications (Backups, diagnostics...) manageable by Jenkins jobs in order to have a web interface for managing your shared projects. You can refer to the documentation for the installation of Jenkins.

The full Jenkins documentation can be found at the following address: https://www.jenkins.io/doc/.

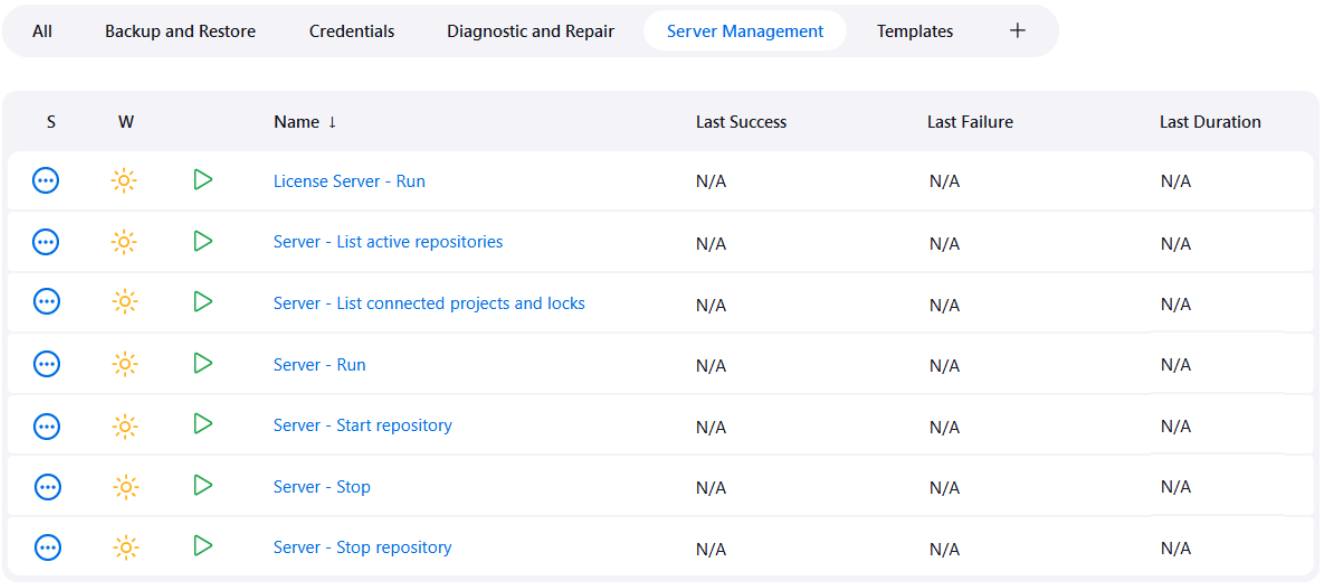
By default, it is available on port **8036**: when logged on the computer running the Scheduler, type the following address in a web browser:

http://localhost:8036

By default, for all jobs, the last 100 job executions (called "builds" in Jenkins) results are kept by Jenkins (build's artifacts and logs). Note that all these jobs can be changed with the Jenkins application.

The default view is the "Server Management" one.

Server Management



Server – List active repositories

This job lists the currently active repositories on the server.

The list result is logged in the console output of the job.

These repositories can be stopped by using the Server – Stop repository job.

Server – List connected projects and locks

This job lists :

- the opened Capella shared projects with the associated username. It corresponds to the CDO sessions opened on the server.
- the currently locked objects classified by opened projects with user information.

This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Server – Run

This job starts the server. By default, this job starts the server every Saturday at 05:15, It never stops (and must not be aborted) except if "Server – Stop" is launched.

Server – Start repository

This job starts a repository on the server, that was previously stopped by the job «Server - Stop repository». When a server starts, all its repositories starts as well.

This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Server – Stop

This job stops the server. By default, this job stops the server every Saturday at 05:00 (and is restarted one hour later by the previous job).

Server – Stop repository

This job stops an active repository on the server.

Use Server – List active repositories to list all active repositories.

The stopped repository cannot be reached, and remote projects existing in this repository cannot be modified. Using the Database – Backup job will not back up the stopped repository.

The server will still be running, and the other non-stopped repositories will still be reachable.

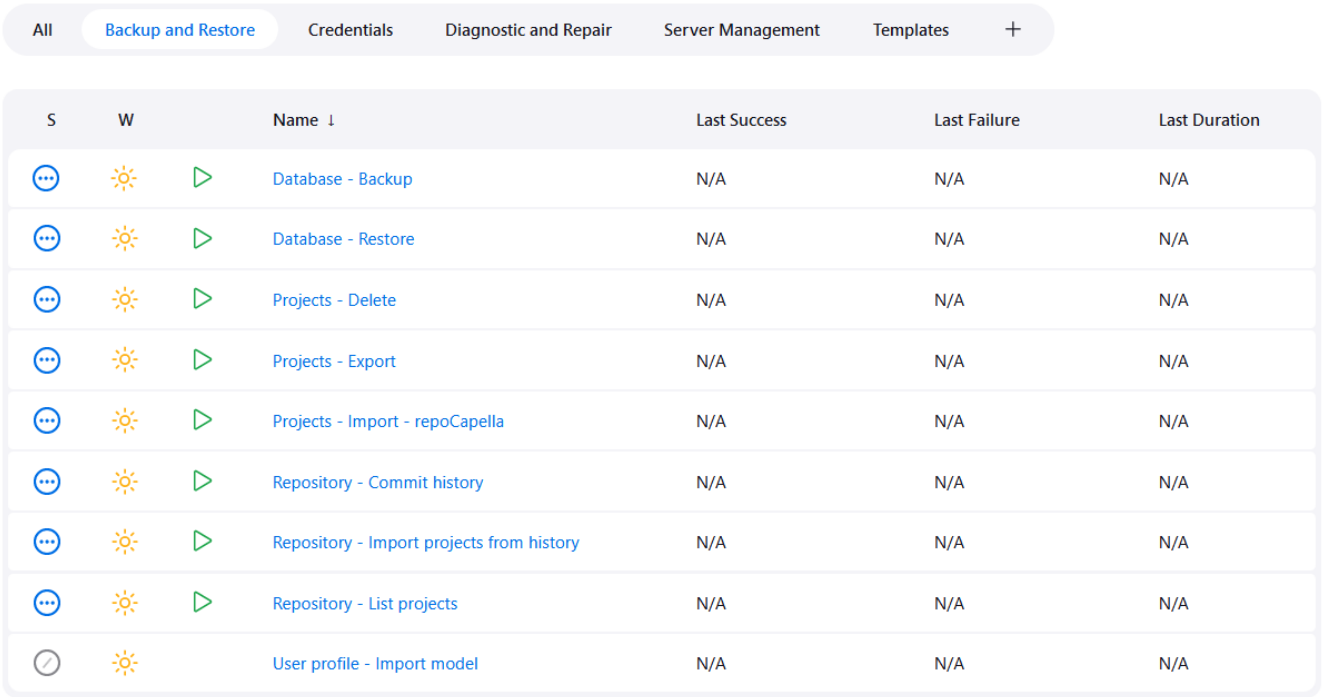
This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

License Server – Run

This job is only present in the commercial versions of Team for Capella.

It allows managing the license server directly from the Scheduler. It is disabled by default.

Backup and Restore



Database – Backup

This job does a dump of the database into a zip file and keeps it as an artifact of the build. By default, it is launched automatically 3 times a day (07:30, 12:30 and 20:30) from Monday to Friday.

Note that this job will perform a backup of the whole server. If several repositories are started, it

creates one zip file per repository.

We strongly recommend having one database path per repository. See How to Add a New Repository

Database – Restore

This job is intended to restore the database from a previously backed-up database.

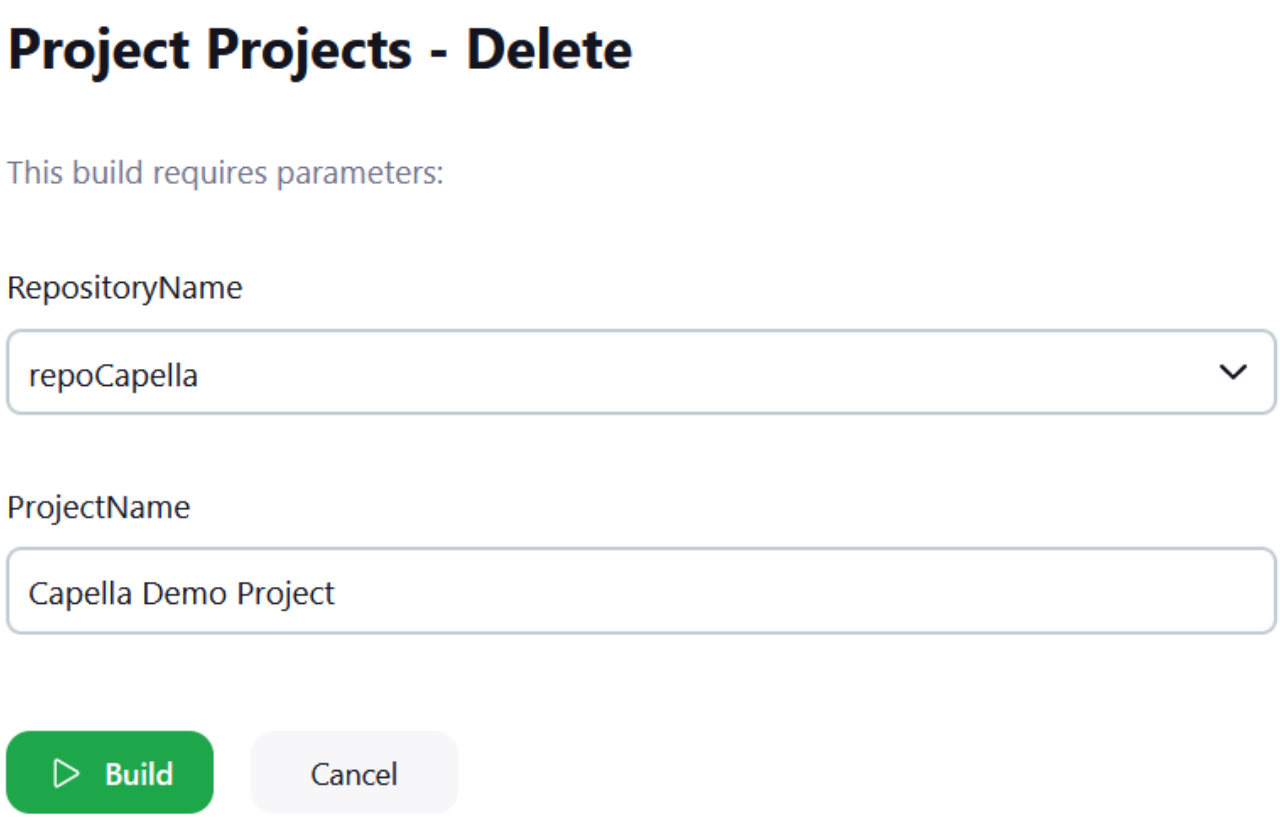
The backup folder is a result of the "Database – Backup" job.

If you want to restore only one repository, move all other archives out of the backup folder to keep the one specific to your repository.

Projects – Delete

It executes the exporter application to delete a project from the given repository without any user interaction.

This job will delete a project according to its name on the server, given as parameter.



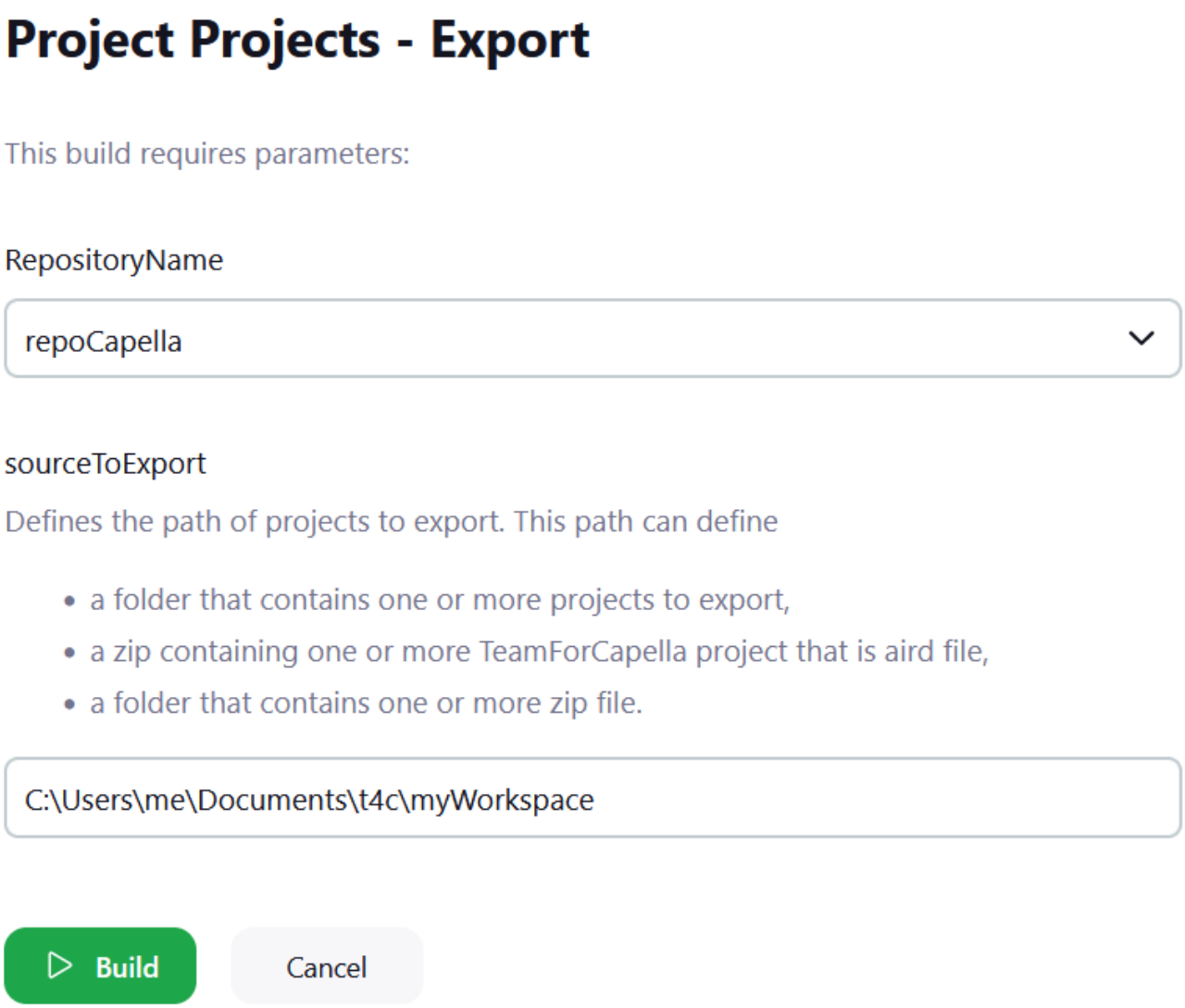
This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Projects – Export

It executes the exporter application to export projects automatically from a local folder (or archive) on the server without any user interaction. This job will export the projects from a specific source. This source can be

- a folder that contains one or more projects to export,
- a zip containing one or more sirius project that is aird file,
- a folder that contains one or more zip files.

This job needs to be configured to specify the folder.



If the job fails, you may have a wrong folder path or none of the representation files have been found in the folder.

This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Projects – Import – repoCapella

It executes the importer application to import projects automatically from a server without any user interaction and archives them as Job's artifacts. By default, it is launched automatically every hour from 07:00 to 21:00 Monday to Friday and targets the default repository (repoCapella).

This job will import the projects for a specific repository. It needs to be configured to specify the repository and optionally, a specific project list to import. If you have many repositories, you ought

to have as many "import projects" jobs that may start at the same time. So you need to configure the number of job executors. Go to Manage Jenkins > configure systems menu if the number of T4C repositories has been extended: # of executors ≥ =nb of repo +3

This job is by default configured to use the *Snapshot import* strategy. Refer to the Importer strategies documentation for more details.

If the job fails, you may have corrupted data in your database that could prevent you from getting imported projects. To avoid further modifications of your projects and minimize data loss, the default behavior in case of import failure is to stop the repository. Then, in order to continue to work on your projects, you may:

- analyze the failure by checking the console log output of the job,
- diagnostic/repair the database with "Diagnostic and Repair" jobs,
- reinitialize the database.

Repository – Commit history

It executes the importer application to import only the commit history.

This job executes with "RepositoryName" as a required parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name. There are additional parameters to this job that can be selected:

- SquashCommitHistory: Squash consecutive commits done by the same user with the same description (default: true);
- IncludeCommitHistoryChanges: Import the commit history detailed changes for each commit done by a user with one of the save actions;
- ComputeImpactedRepresentationsForCommitHistoryChanges: Compute the impacted representations while exporting changes.

See importer documentation in Importer Parameters chapter for more information about these parameters.

Repository – Import projects from history

It executes the importer application to import projects of a repository at a given date.

This job executes with required parameters:

- RepositoryName: the name of the repository to import from;
- ProjectName: name of the project to import. By default the value is "*" in order to import all project of the repository;
- CheckoutTimestamp: reference date to import from. By default the value is "HEAD" to import from the latest available commit. Otherwise, the date pattern is yyyy-MM-ddThh:mm:ss.SSSZ. See full documentation about the timestamp in Importer Parameters Notes

See importer documentation in Importer Parameters chapter for more information about these

parameters.

Repository – List projects

This jobs list all available project in a given repository.

This job executes with required parameters:

- RepositoryName: the name of the repository to import from;
- CheckoutTimestamp: reference date to import from. By default the value is "HEAD" to import from the latest available commit. Otherwise, the date pattern is yyyy-MM-ddThh:mm:ss.SSSZ. See full documentation about the timestamp in Importer Parameters Notes

See importer documentation in Importer Parameters chapter for more information about these parameters.

User profile – Import model

This jobs extracts the user profile model from the database and saves it locally in the *outputFolder*.

It is disabled by default and must be enabled only if the repository is configured to use the "User Profiles" access control mode.

Diagnostic and Repair

These jobs cannot be started if the authenticator is based on an OpenID Connect. You must start the server with another mode of authentication or no authentication.

Repository – Diagnostic

This maintenance job needs to be manually launched. This job runs a diagnostic to detect inconsistencies described in *Server Administration / Administration Tools / Repository maintenance application*.

The diagnostic result is logged in the console output of the job. It is kept as an artifact of the job result.

The diagnostic is run for a specific repository and need to be configured according to your repository name.

This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Repository – Maintenance

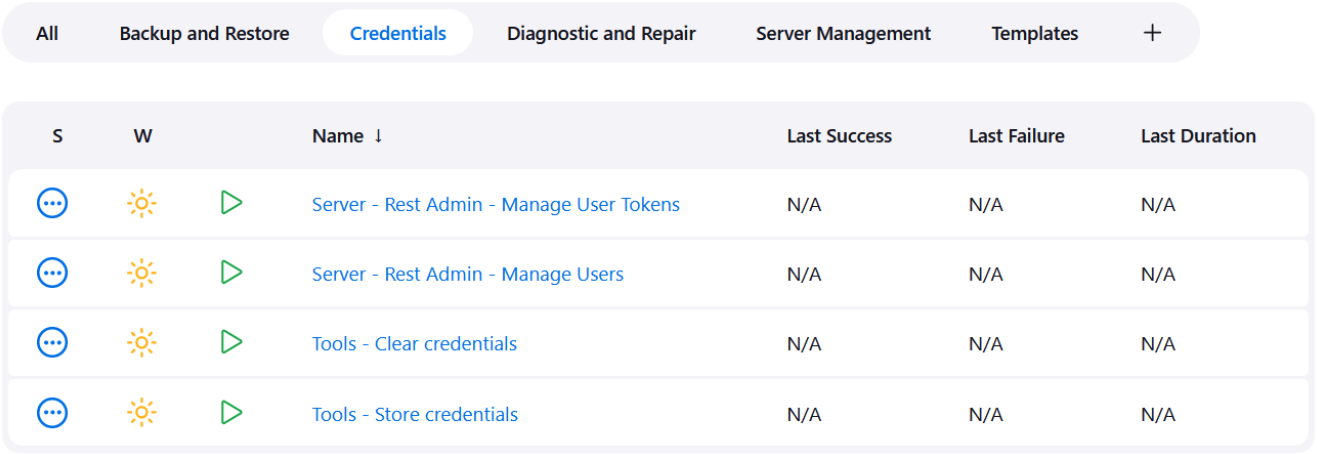
This maintenance job needs to be manually launched. It is recommended to launch the Repository – diagnostic job first.

It runs a diagnostic in order to detect inconsistencies described in *Server Administration / Administration Tools / Repository maintenance application*. Then, it launches the maintenance tasks if some managed issues are detected: it will back up the server with capella_db command, perform the required changes on the database and close the server. The steps are logged in the console output of the job, and the corresponding log file is kept as an artifact of the job result.

The maintenance is run for a specific repository and needs to be configured according to your repository name.

This job executes with "RepositoryName" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

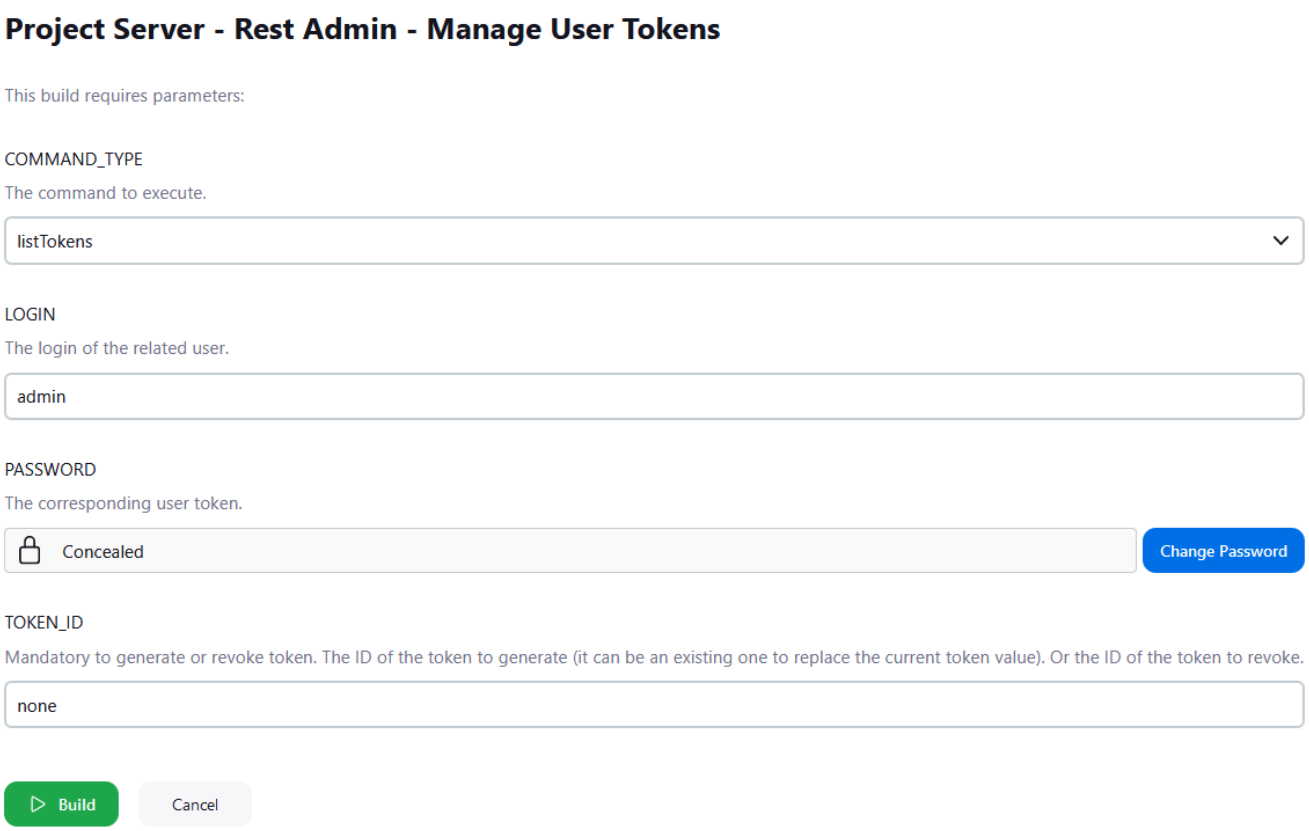
Credentials



Server – Rest Admin – Manage User Tokens

This jobs executes the Tools Credentials Application to manage the access tokens to the Rest API for a specific user.

Launching a build requires setting values for four parameters:

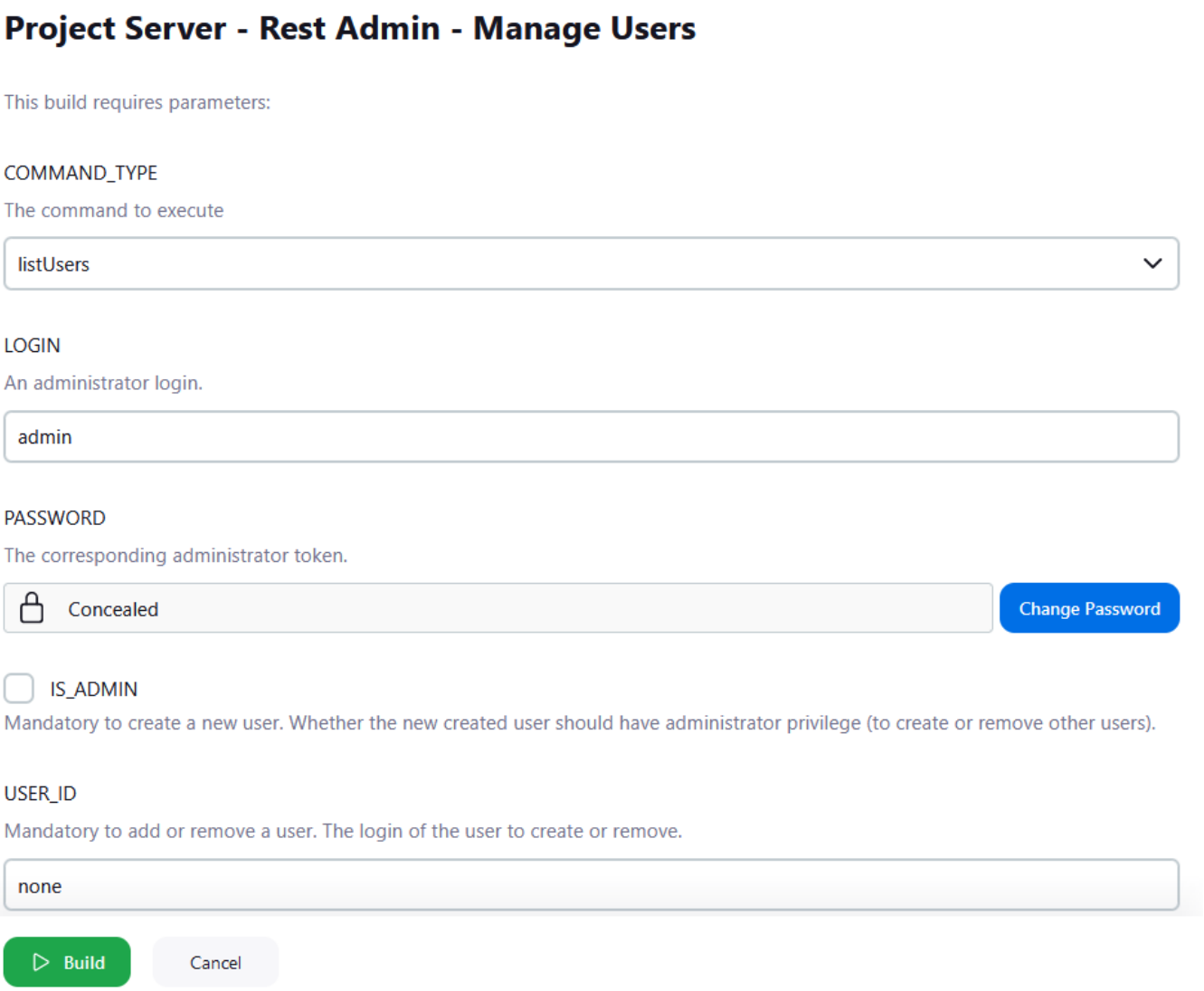


Note that the login and password must be valid credentials for the REST Admin API.

Server – Rest Admin – Manage Users

This jobs executes the Tools Credentials Application to manage the Rest API registered users.

Launching a build requires setting values for five parameters:

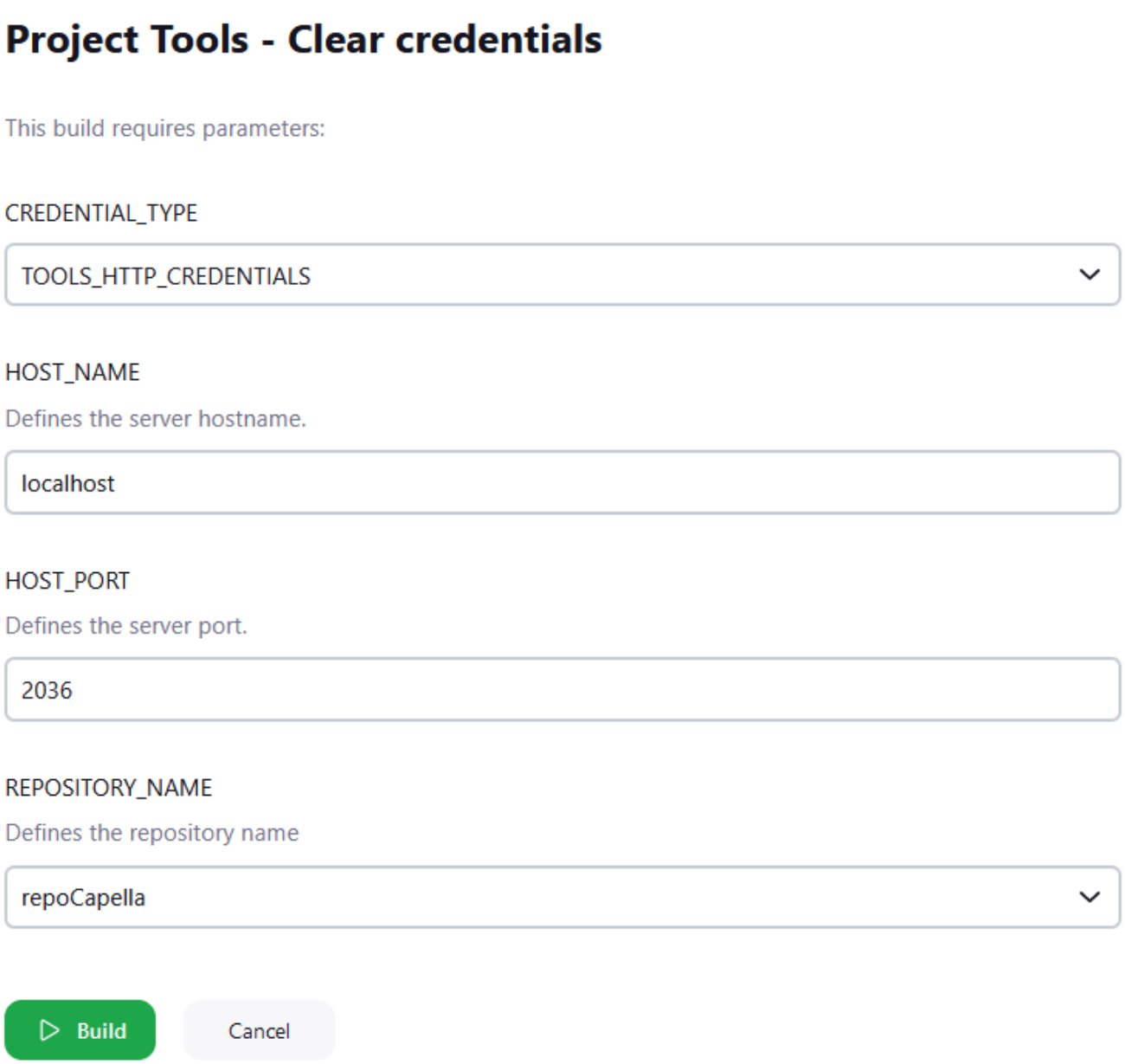


Note that the login and password must be valid credentials for the REST Admin API.

Tools – Clear credentials

This job executes the credentials application to clear credentials in Eclipse Secure Storage, allowing the importer application to connect to the rest admin server or to connect to a CDO repository.

As credentials needs to be associated with a repository, when this job is executed, it will start by asking to fill the following parameters:



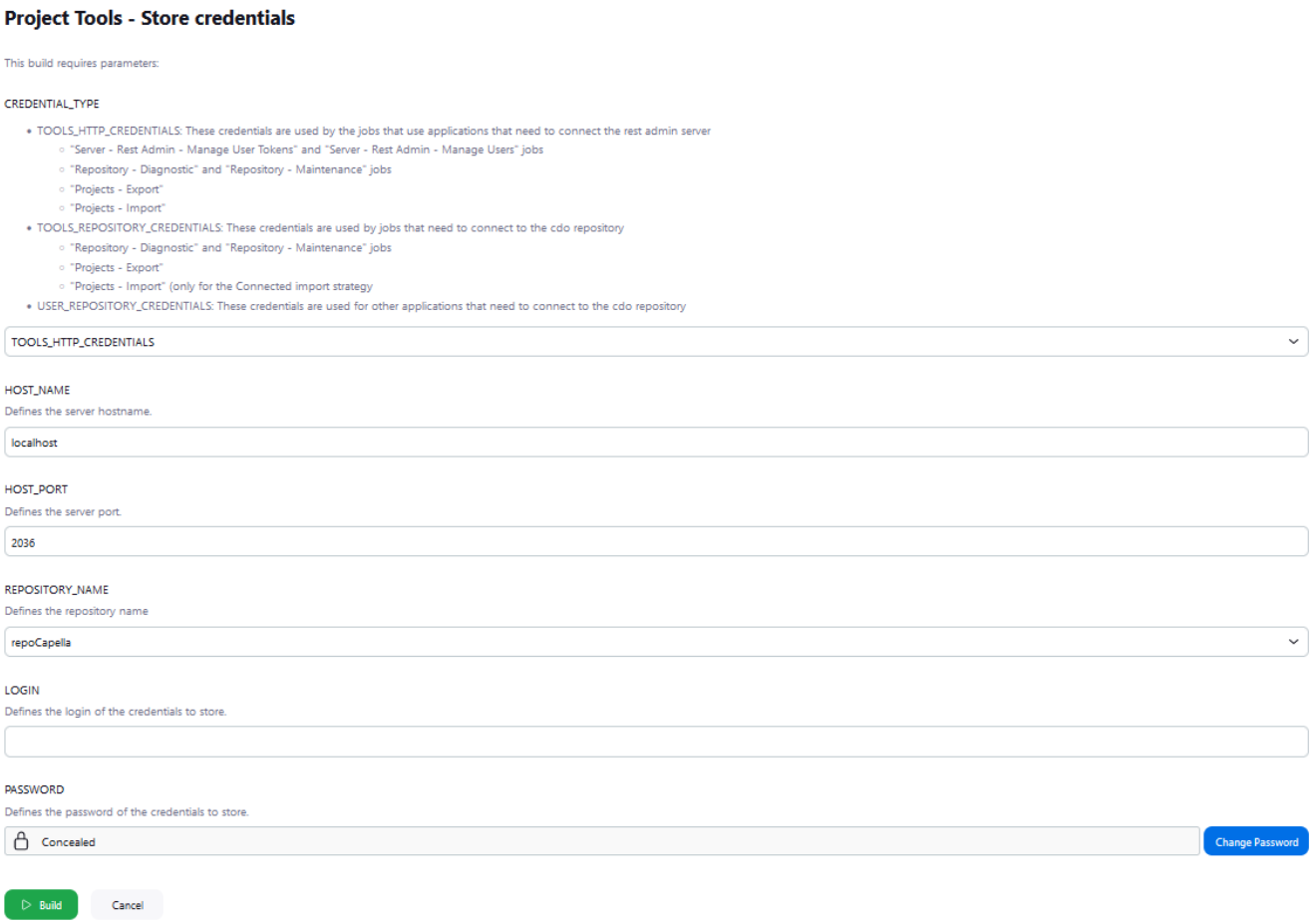
Note that credentials are required only with the *Connected import* strategy. See Importer strategies for more details.

This job executes with "REPOSITORY_NAME" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Tools – Store credentials

This job is the opposite of the previous one, it stores the credentials in Eclipse Secure Storage, allowing either to connect to the rest admin server or to connect to a CDO repository.

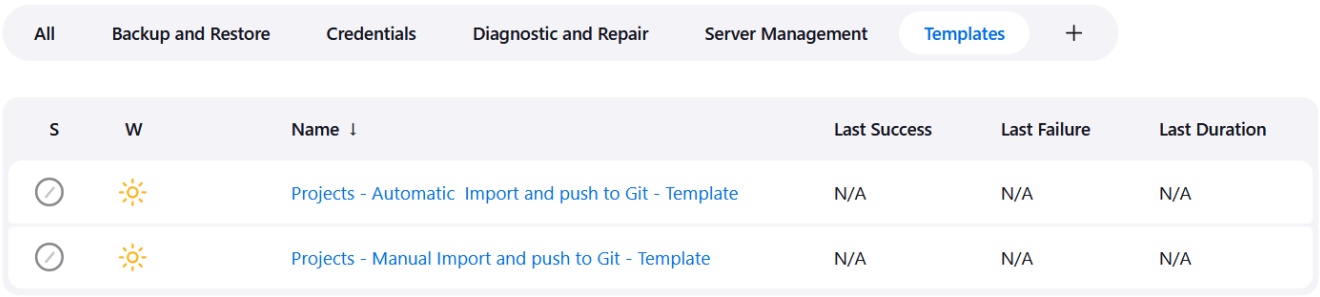
As credentials needs to be associated with a repository, when this job is executed, it will start by asking to fill the following parameters:



Note that credentials are required only with the *Connected import* strategy. See Importer strategies for more details.

This job executes with "REPOSITORY_NAME" as a parameter. As it is a list with only the default "repoCapella" value, it can be edited if your server configuration has more repositories or with a different name.

Templates



This view contains templates of jobs which are disabled by default. They are provided as an example to show how to create backup jobs whose result is pushed to a Git repository.

See each job description in the Scheduler to see how to use them.

How to Start the Team for Capella Scheduler

The Jenkins installation should have included the creation of a new service (named Jenkins) that

automatically starts Jenkins with the system.

Windows

If you do not have the Jenkins service, go to Jenkins (or start it manually from its installation folder), go to the **Manage Jenkins** configuration page and select **Install as a Windows service**.

Linux

The Jenkins service can be started or stopped by using the **systemctl** command:

```
systemctl start jenkins
```

How to start the Server when Scheduler starts

To start the Team for Capella Server automatically when the scheduler starts (i.e.: launch the Start server job), go to the configuration page of the Start server job and then check the box "Build when job nodes start", the "Quiet period" parameter allows delaying the start:

How to change job scheduling

Every job contains in its configuration page a text field called "Schedule". Use this field to change the Job's scheduling configuration. It is visible on the previous screenshot.

How to Stop the Team for Capella Scheduler

To stop the Jenkins scheduler, go to the **Manage Jenkins** page and select **Prepare for Shutdown**

Prepare for Shutdown

Stops executing new builds, so that the system can be eventually shut down safely.

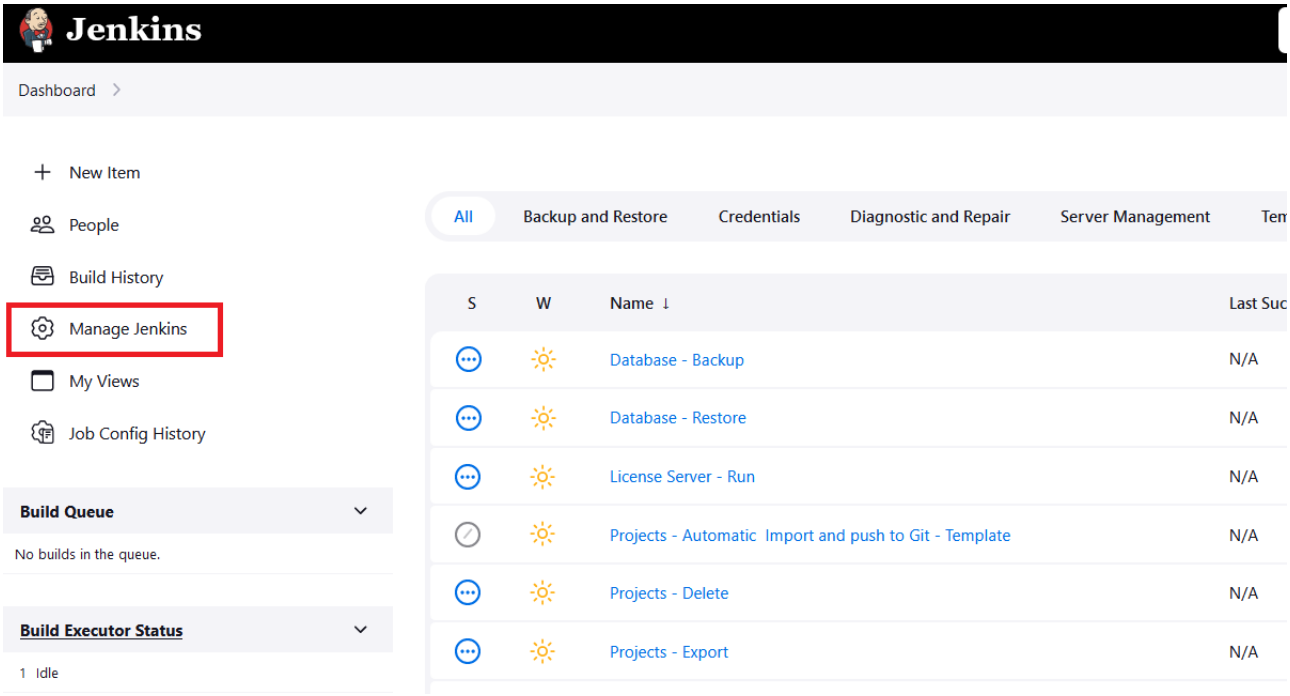
This allows sending a warning to anyone currently connected to the scheduler and end the jobs currently running or in queue. After that, you can simply go to the Windows services and stop the **Jenkins** service.

Activate Security in Jenkins

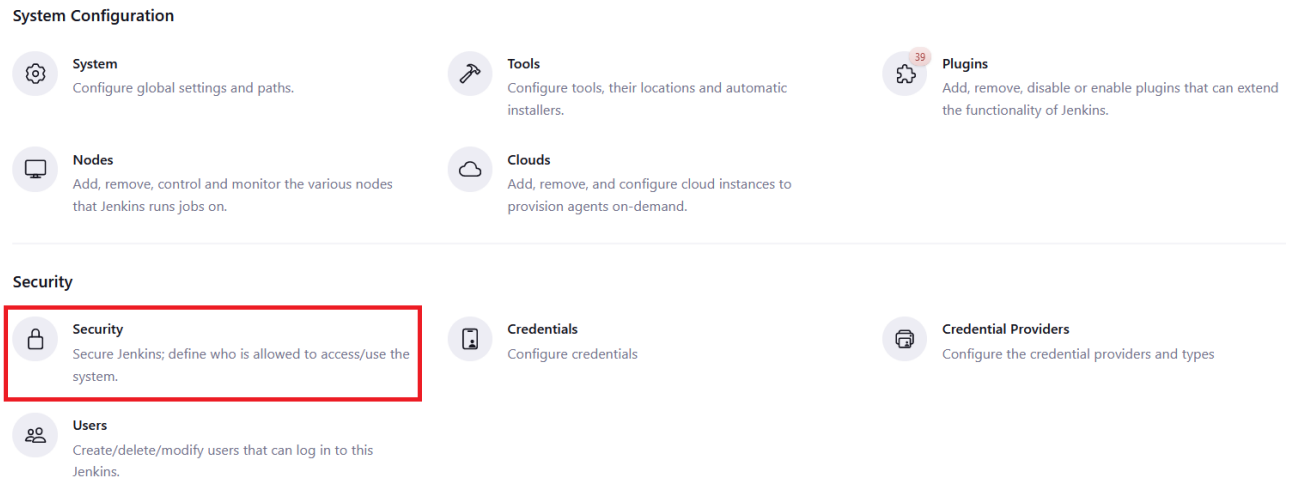
By default, in the scheduler, the security checks are disabled. This means that Jenkins is available to anyone who can access Jenkins web UI without asking for their login and password.

It is possible to configure security within Jenkins to define a group of users, which are allowed to log in to Jenkins or to check user passwords against the username in LDAP or in Jenkins' own user database. To do that, the procedure is the following:

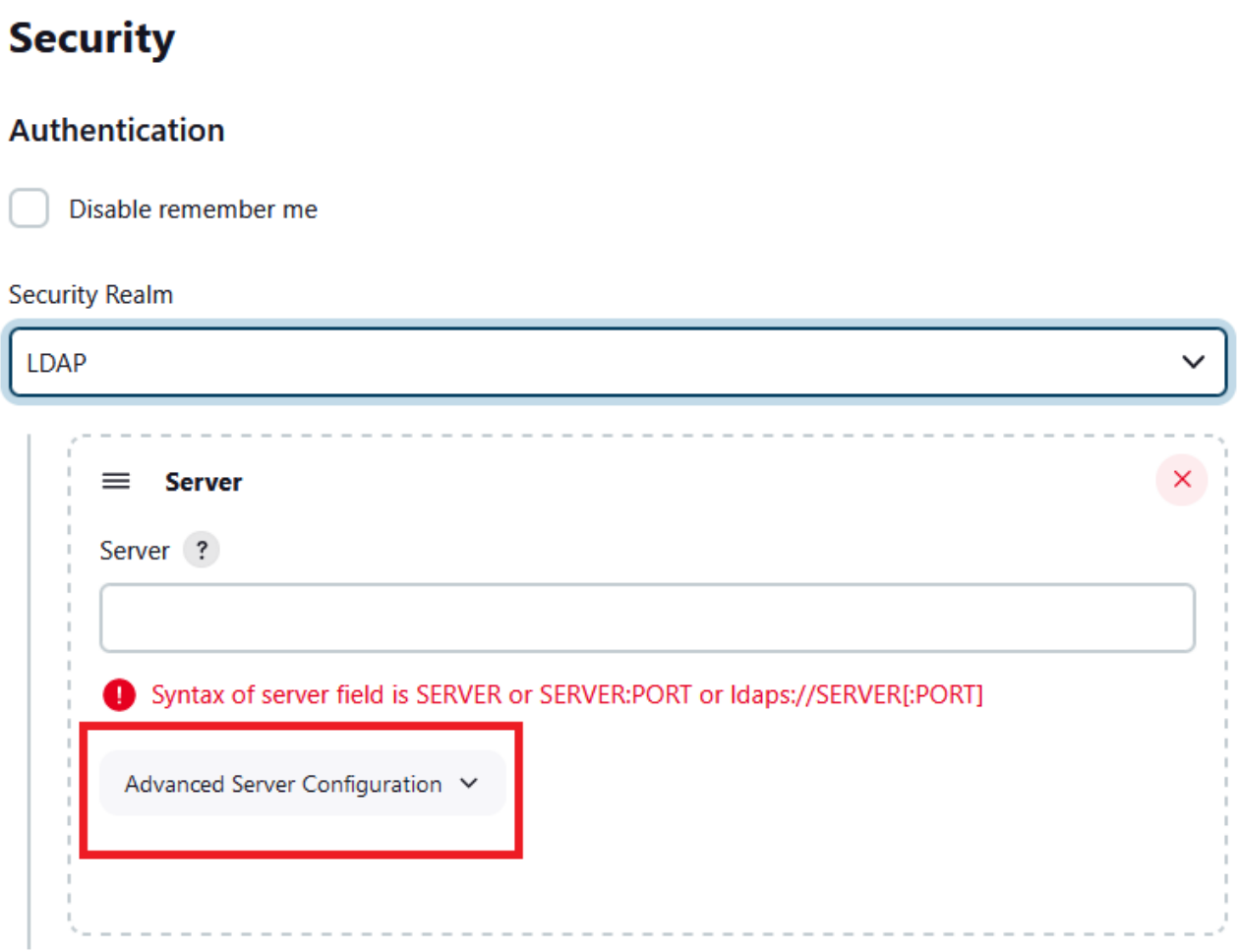
1. Connect to Jenkins as a user with administration rights.
2. Select ***Manage Jenkins***



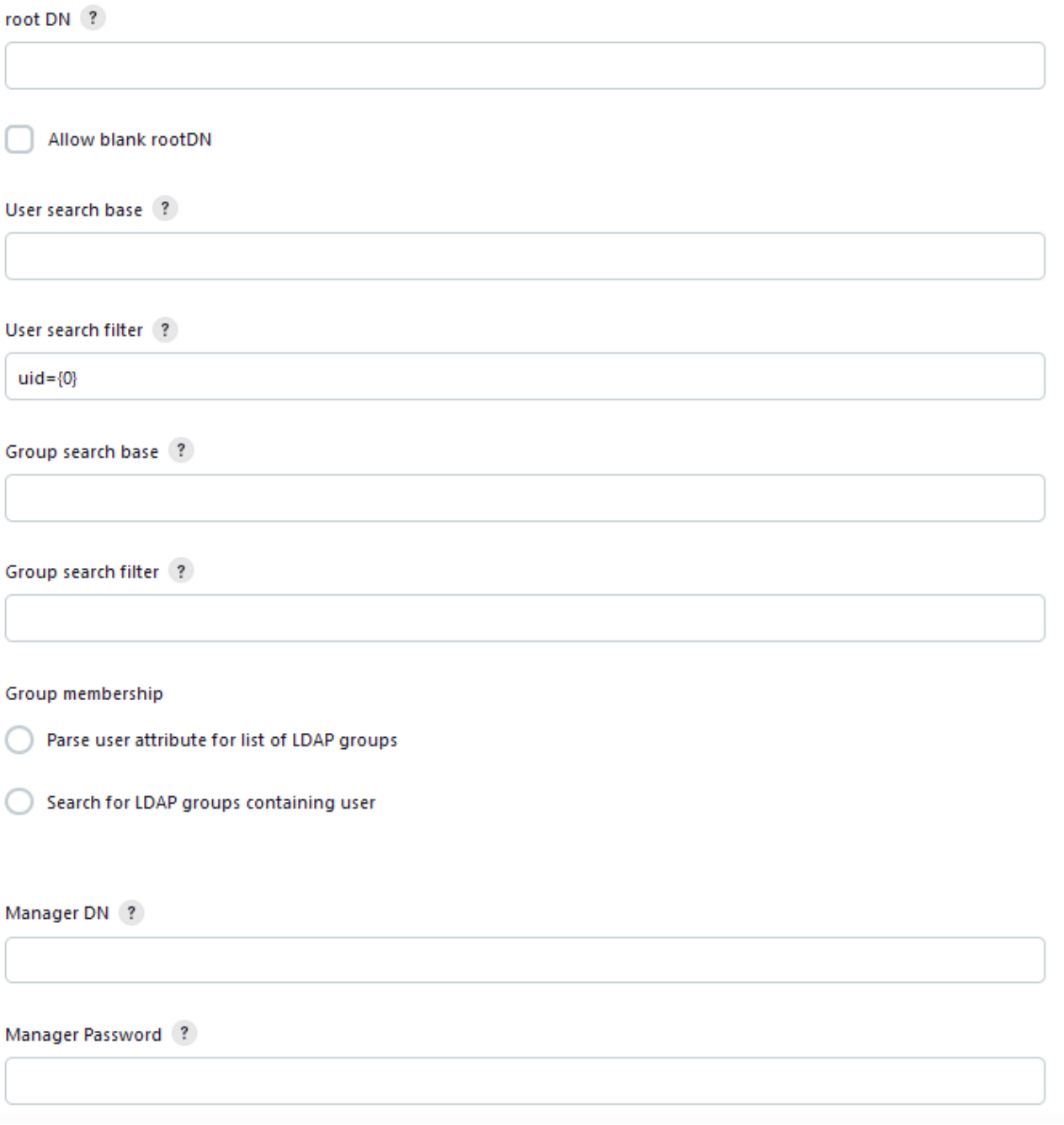
3. Select ***Configure Global Security***.



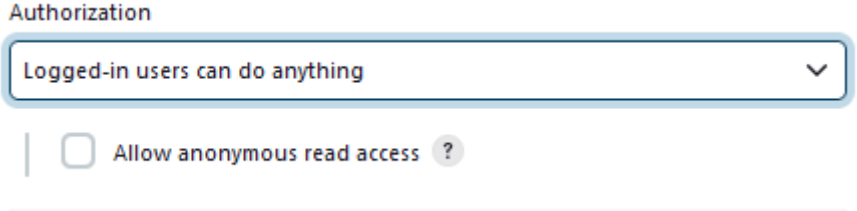
4. Select the ***Jenkins' own user database*** security realm radio button to register users in Jenkins or select the ***LDAP*** radio button to register configurations for the LDAP servers that Jenkins should search.
5. To configure an LDAP server, select the corresponding radio button and then the ***Advanced...*** button underneath the Server text field.



6. Enter the LDAP settings as shown in the following diagram:



7. **Note:** The group specified in ***Group search base*** and the username specified in ***Manager DN*** may need to be changed. The password specified in ***Manager Password*** is the password for the user in the ***Manager DN*** field.
8. To ensure that only logged-in users can perform actions, select **Authorization** → **Logged-in users can do anything**.



9. Save the configuration changes.
10. Log in to Jenkins via the **log in** link in the top right-hand corner of the screen.

You can also decide to use the Jenkins' own user database:

1. Connect to Jenkins as a user with administration rights.
2. Select ***Manage Jenkins***.
3. Select ***Configure Global Security***.
4. Select the ***Enable security*** checkbox, the ***Jenkins' own user database*** security realm radio button and then place a check mark next to ***Allow users to sign up***.
5. Save
6. Create a user (menu in the top-right corner)
7. Log in to Jenkins via the **log in** link in the top right-hand corner of the screen and go back to http://localhost:8036/configure (or select ***Manage Jenkins*** and then ***Configure Global Security***).
8. In the security realm section, remove the check mark next to ***Allow users to sign up***
9. In the ***Authorization*** section, select the ***Matrix-based security*** mode,
10. In the text box below the matrix, type your username and click ***Add***
11. Give yourself full access by checking the entire row for your username
12. Configure other users
    - Repeat the two previous steps for other users who deserve full access.
    - If you want to allow anonymous users to see the jobs: Give the Anonymous user only Overall Read access.
    - You can also decide to create specific users who can only launch the jobs and see the results and hide everything for anonymous users.
13. Click Save at the bottom of the page. You will be taken back to the top page.
14. Restart Jenkins

More details can be found in *https://www.jenkins.io/doc/book/system-administration/security/*.

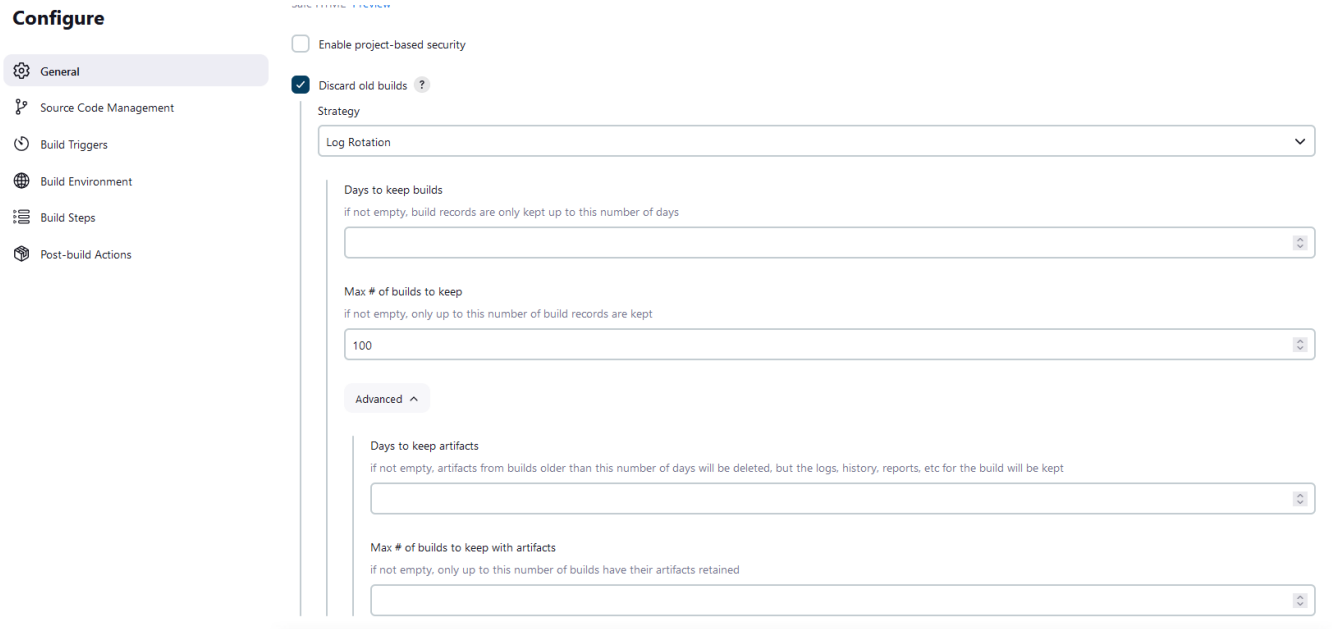
Microsoft Entra ID authentication for Jenkins

A Jenkins plugin allows the authentication to be handled by Microsoft Entra ID. This plugin is automatically installed by the Jenkins plugins for Team for Capella installation script, but if you have installed Jenkins by another mean, it can be installed as follows: First, go to **Manage Jenkins** > **Manage Plugins**. On the **Available** tab, look for **Microsoft Entra ID Plugin**. Before installing it, hover your mouse over the label and open the link on a new tab. This will open a documentation page useful later. Now, check the plugin and press the download and install button. Restart Jenkins. Once restarted, Jenkins is ready to be configured for an authentication with Microsoft Entra ID. For that, go to the tab that was opened previously and follow the documentation. There are two parts for this configuration, one in Microsoft Entra ID and one in Jenkins. Note that on the Jenkins setting

part, when asked to fill the **Tenant** this corresponds to the **Directory (tenant) ID** in your Entra ID application. It is not necessarily the same value as in the CDO server configuration files (for instance, the value "organizations" can be used instead of Tenant ID for the purpose of OpenID discovery mechanism). Also, a test user is asked in order to verify the authentication parameters. This is not the name that is needed here but the **User Principal Name** or the **Object ID** of this user. Note that, if you want to have a different list of users having access to Jenkins (compared to the users that have access to the CDO server), you can create a new application on Entra ID dedicated to the scheduler access (Jenkins).

How to Change Backup and Import Files Purge Policy

- Connect to the scheduler admin site
- Select the "Database – Backup" job → Configure
- In the section "Delete old builds" → Update the maximum number of build to keep and the max # of builds to keep with artefact



- Select the "Projects – Import" job → Configure
- Update the section "Delete old builds" like in the step 3)

How to Dissociate Multiple Projects in Jenkins

Purpose

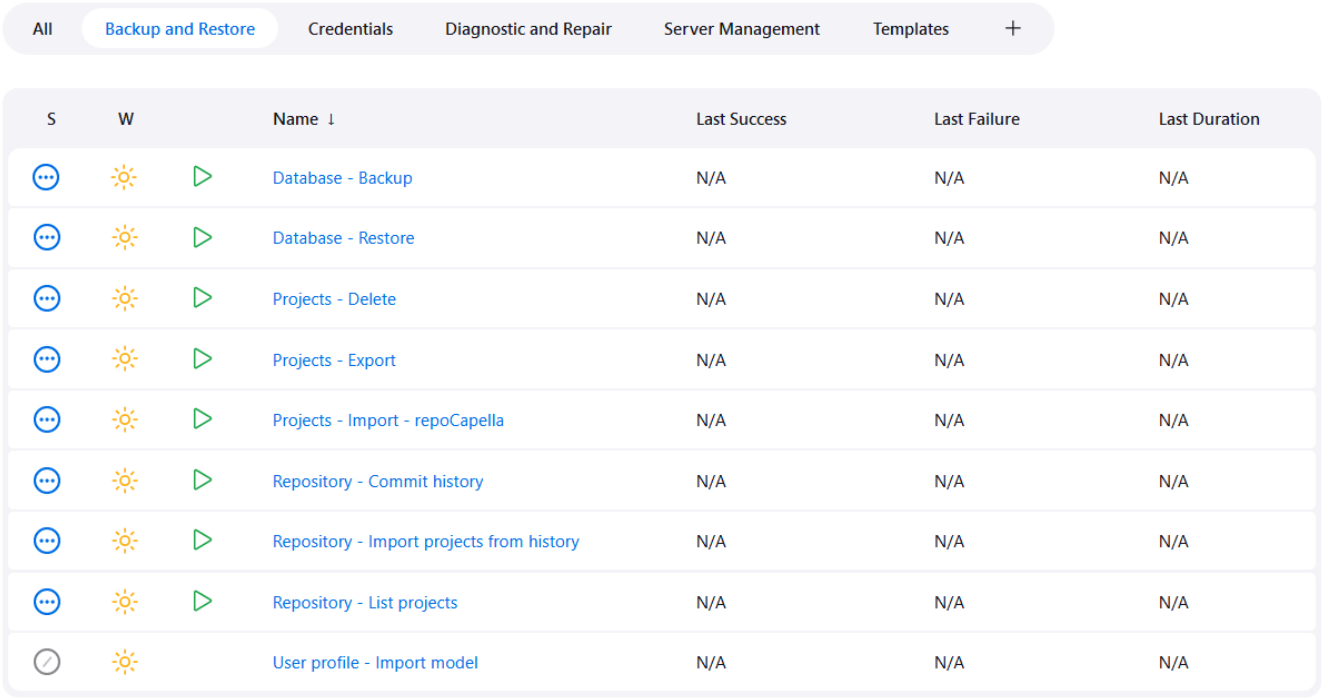
I have 2 modeling projects (or more) working with Team for Capella and I want to isolate them in Jenkins (a person logged in Jenkins must see only Jenkins jobs dedicated to its project).

The proposed solution uses the internal Jenkins user database but is applicable with some changes to use a LDAP server.

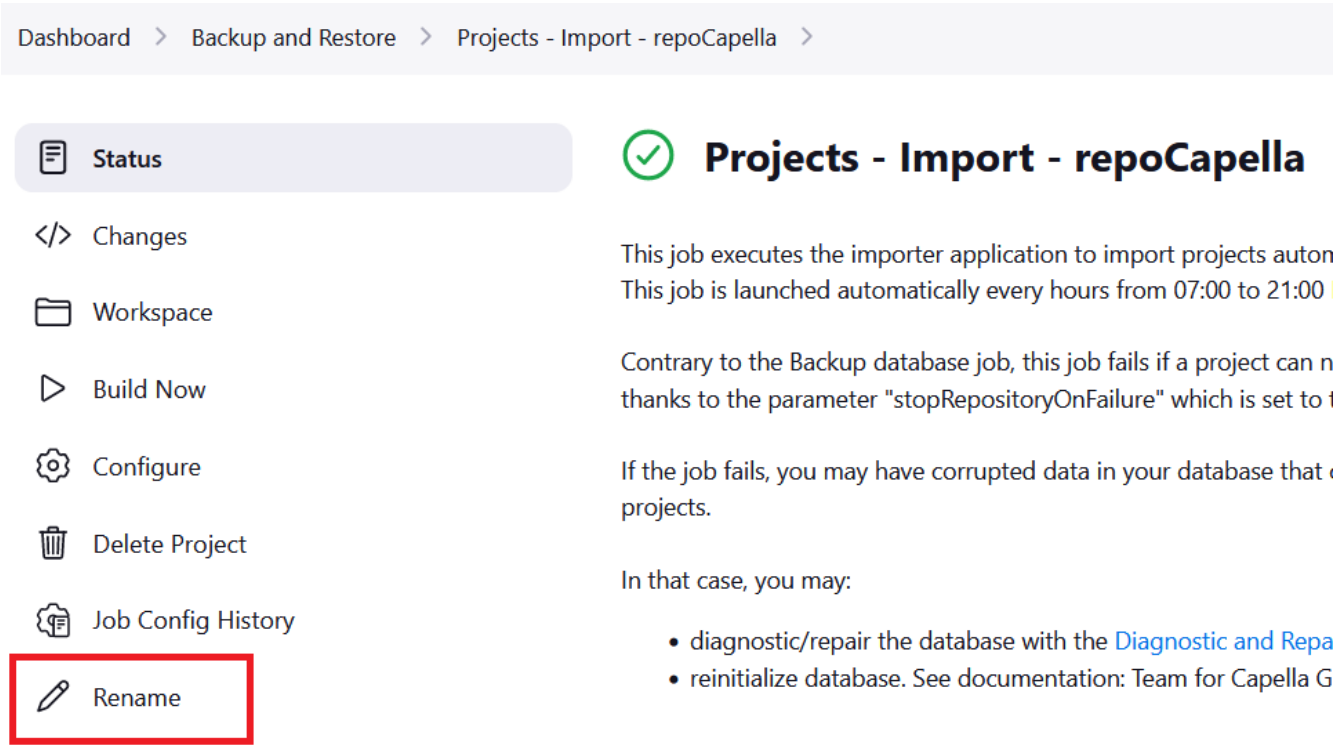
Note that this section be adapted for different situations: multiple projects, multiple repositories or even multiple servers managed yby the same Scheduler.

Jobs Creation

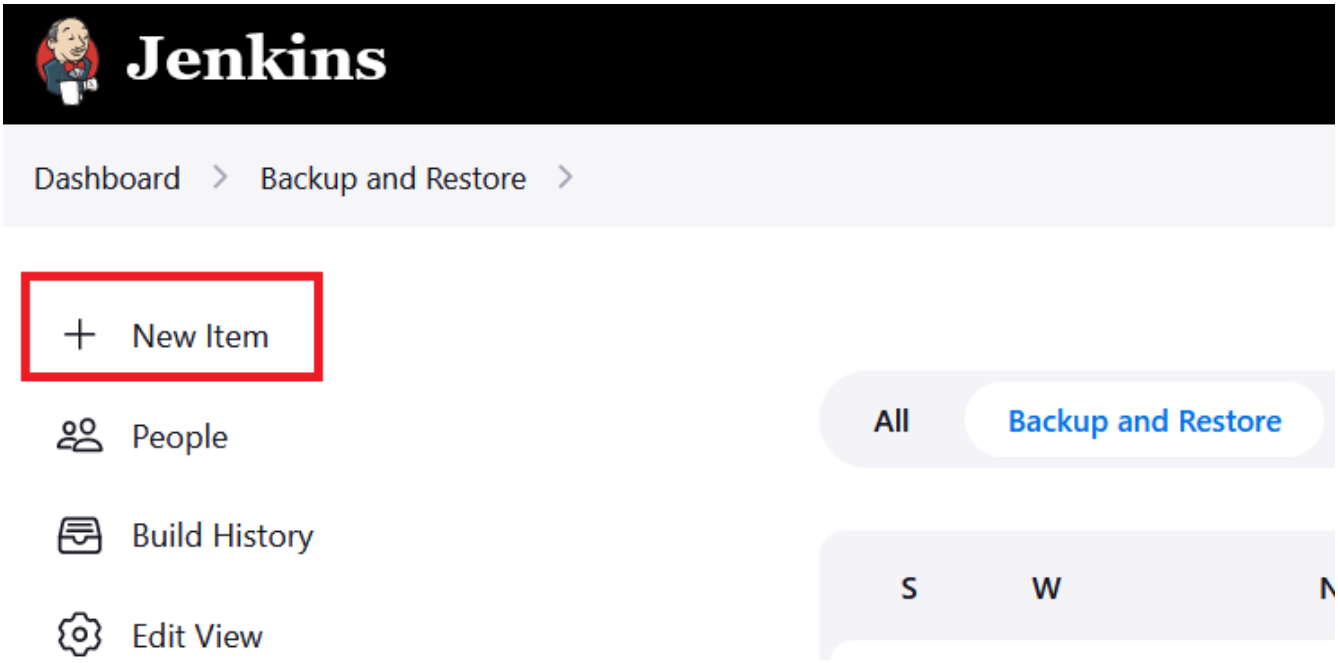
When Jenkins is started for the first time, it contains all necessary jobs:



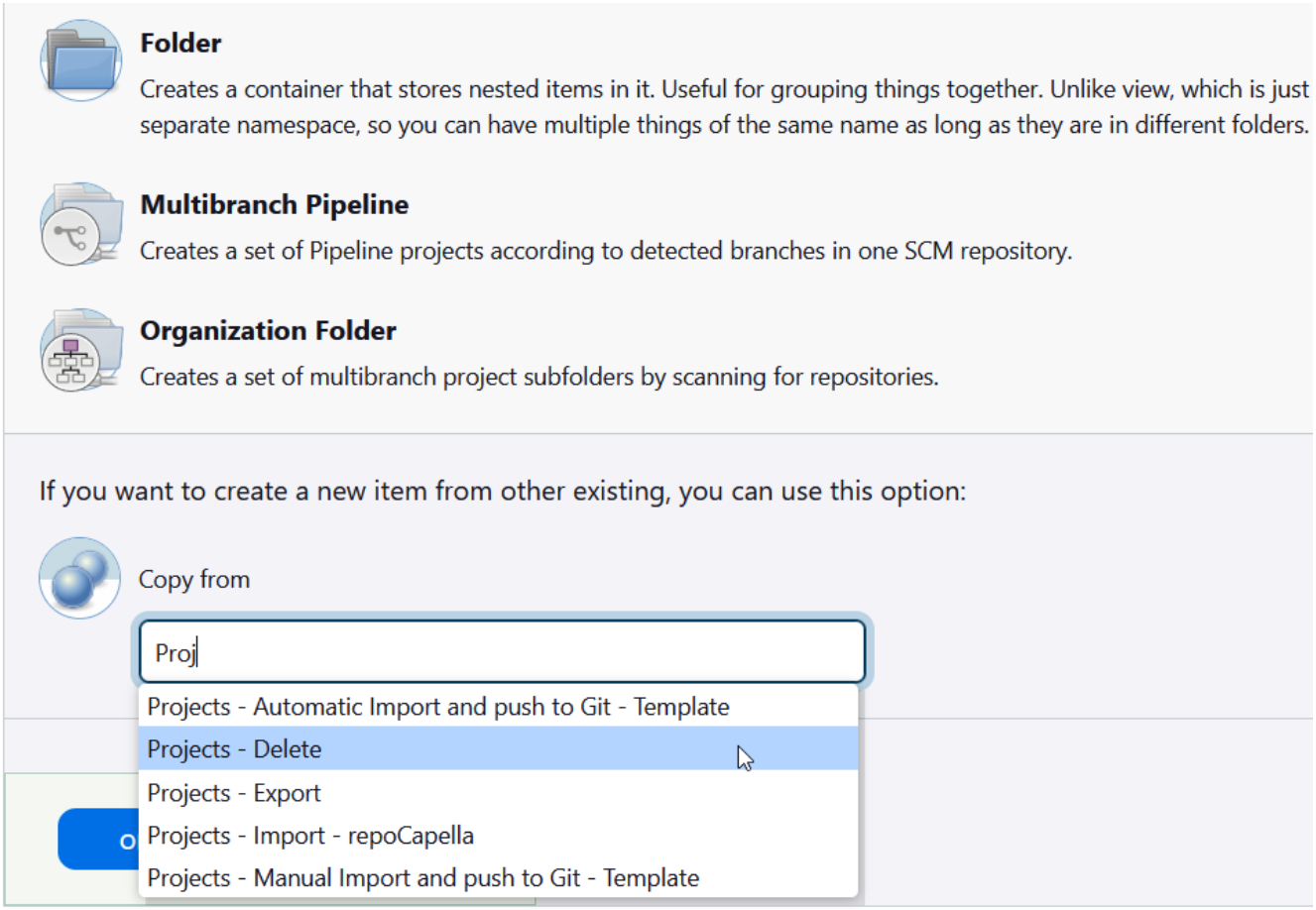
Let's say the "Projects – Import" job will be used for Project 1. So, rename it to "Project 1 – Import":



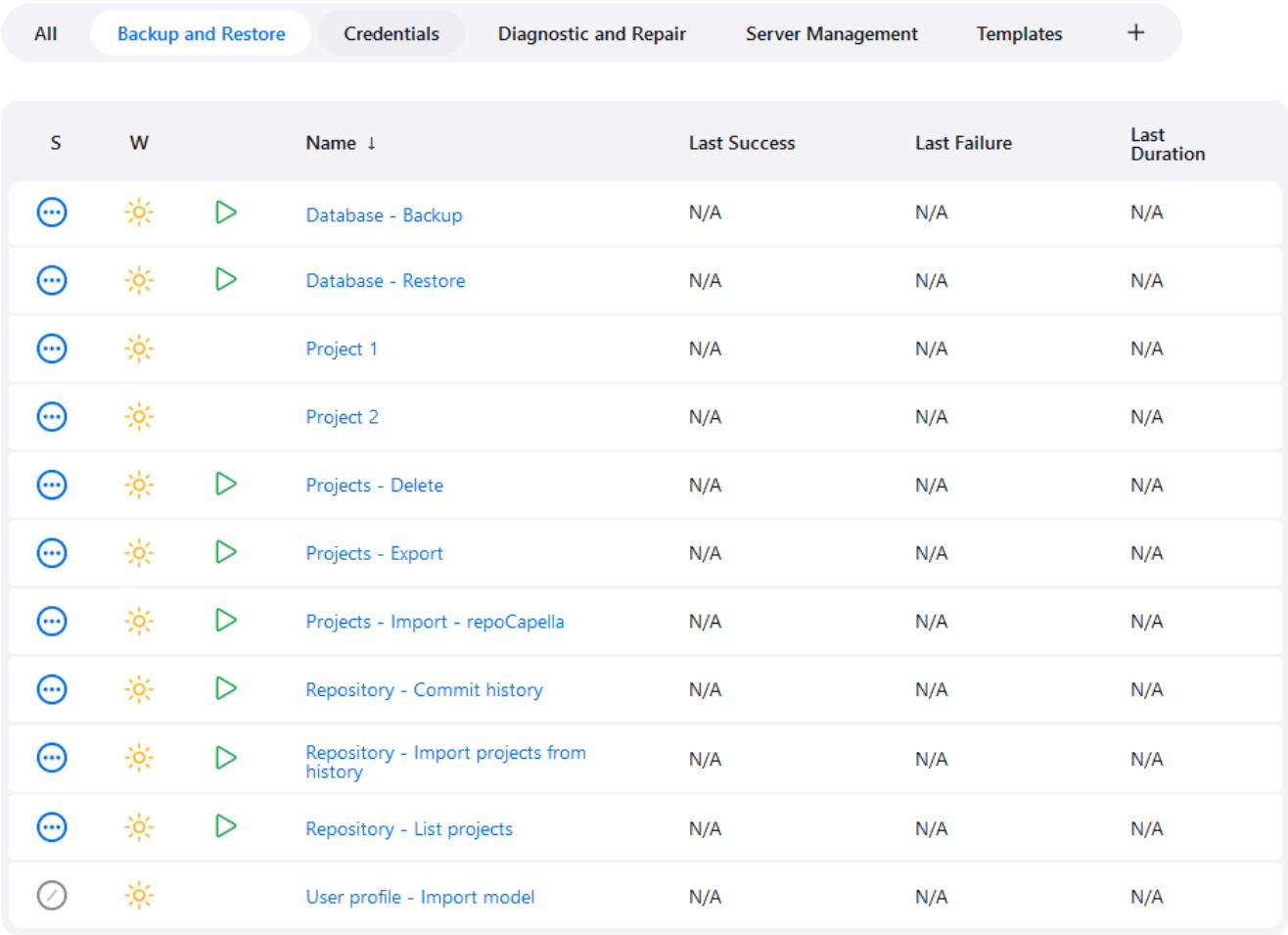
Now we will create jobs for Project 2. Click on the "New Item" in the "Backup and Restore" tab.



Then select "Copy existing Job"). Copy the "Project 1 – Import" job and rename it into "Project 2 – Import".



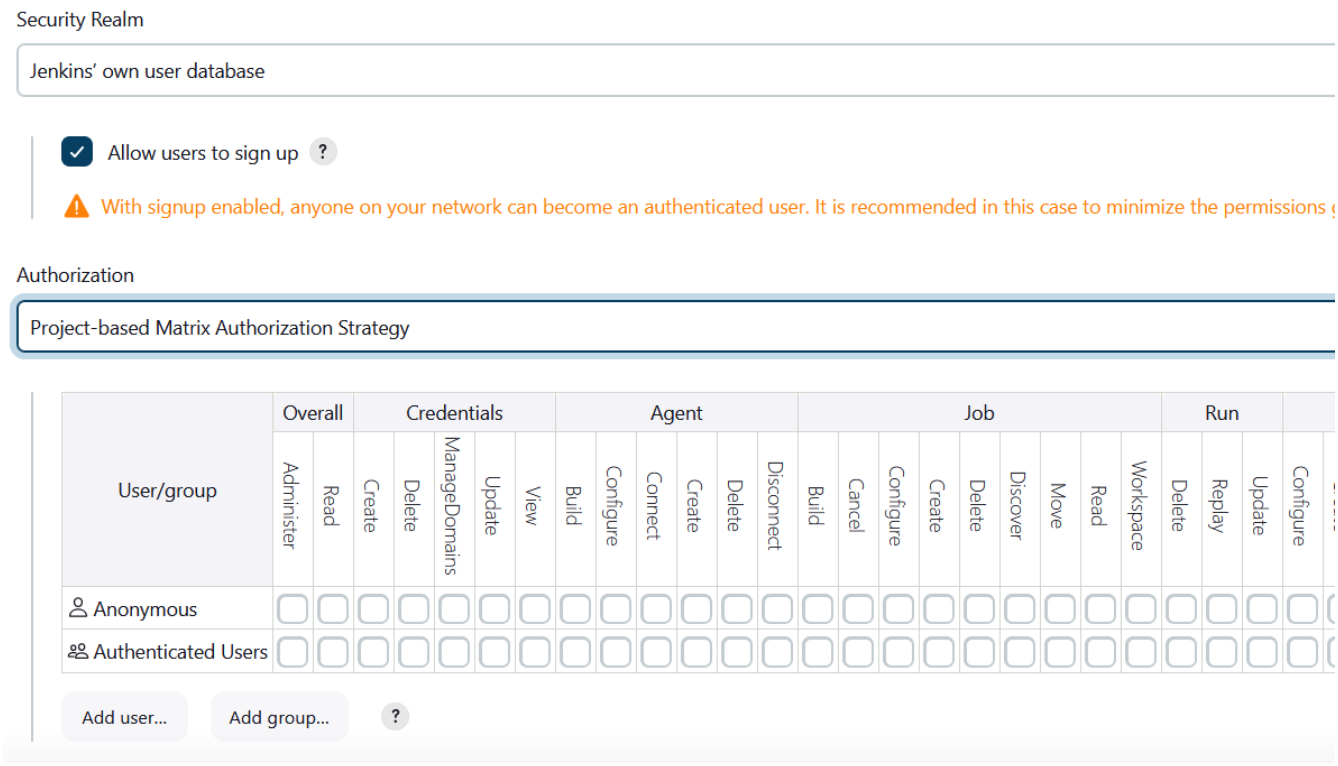
The result is the following:



Project 1 and Project 2 jobs have to be configured correctly to be used (their build step must be modified to add -projectName ProjectXName) and number of executors has to be increased.

Access Rights Definition (whole Jenkins instance level)

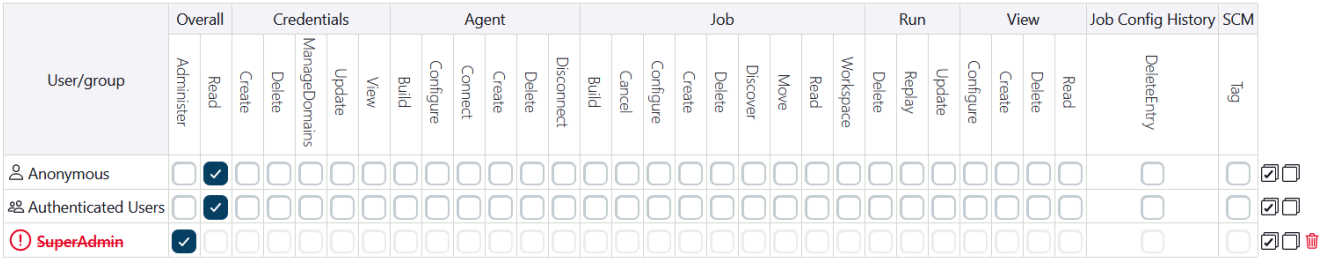
Go to "Manage Jenkins" / "Configure Global Security", set parameters as shown in the screenshot:



Do the following changes in the table:

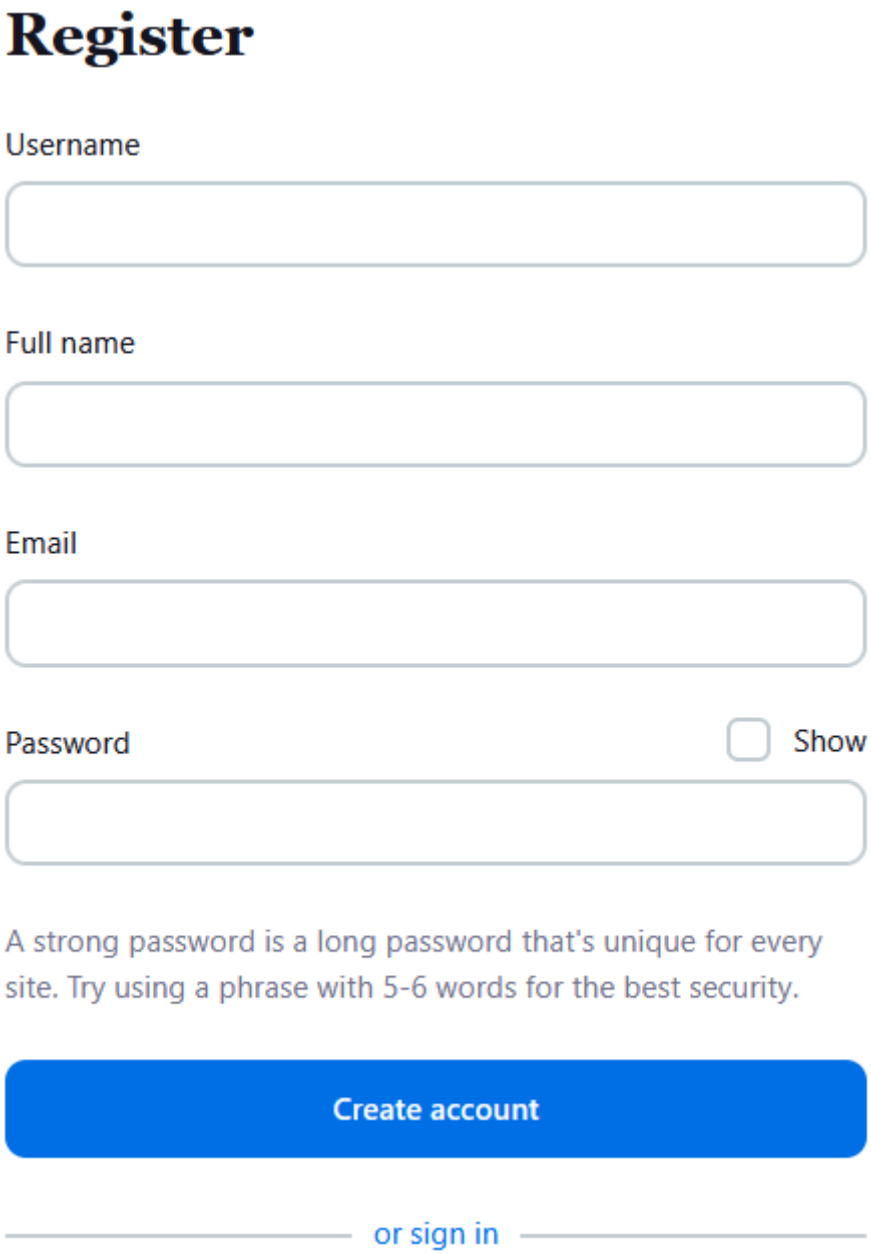
- For "Anonymous Users": check the "Overall" / "Read" check box if anonymous access can be granted,
- For "Authenticated Users": check the "Overall" / "Read" check box,
- Add a "SuperAdmin" user and give it all rights by checking all check boxes,

The table must be as follows:



Click on "Save".

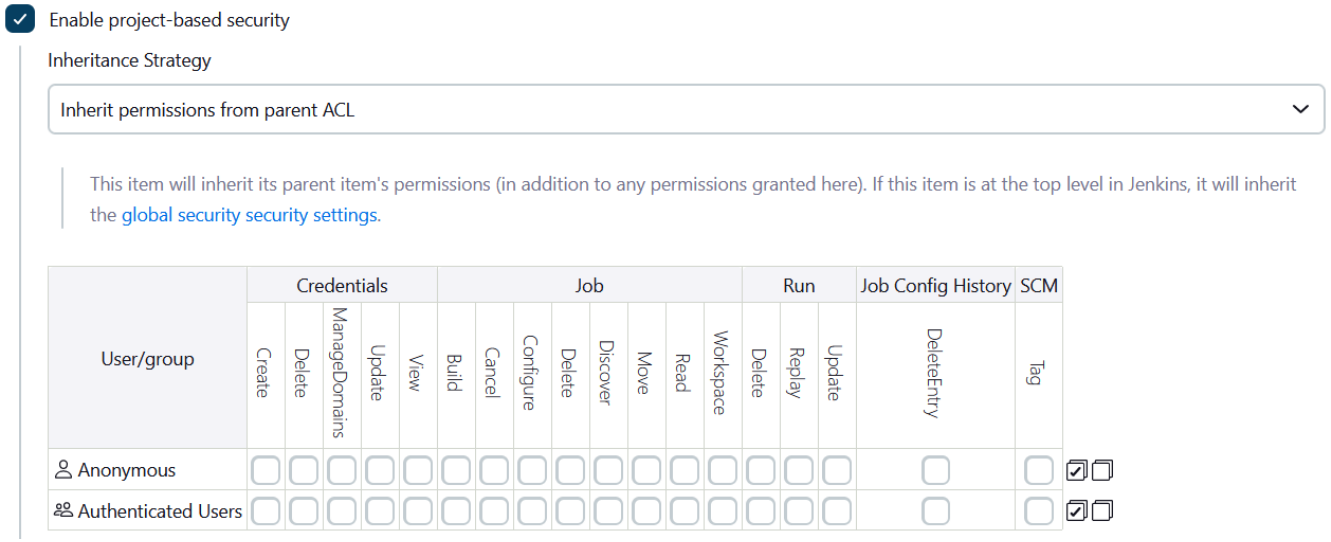
Access rights are now activated:



Create the "SuperAdmin" account and use it to log in Jenkins.

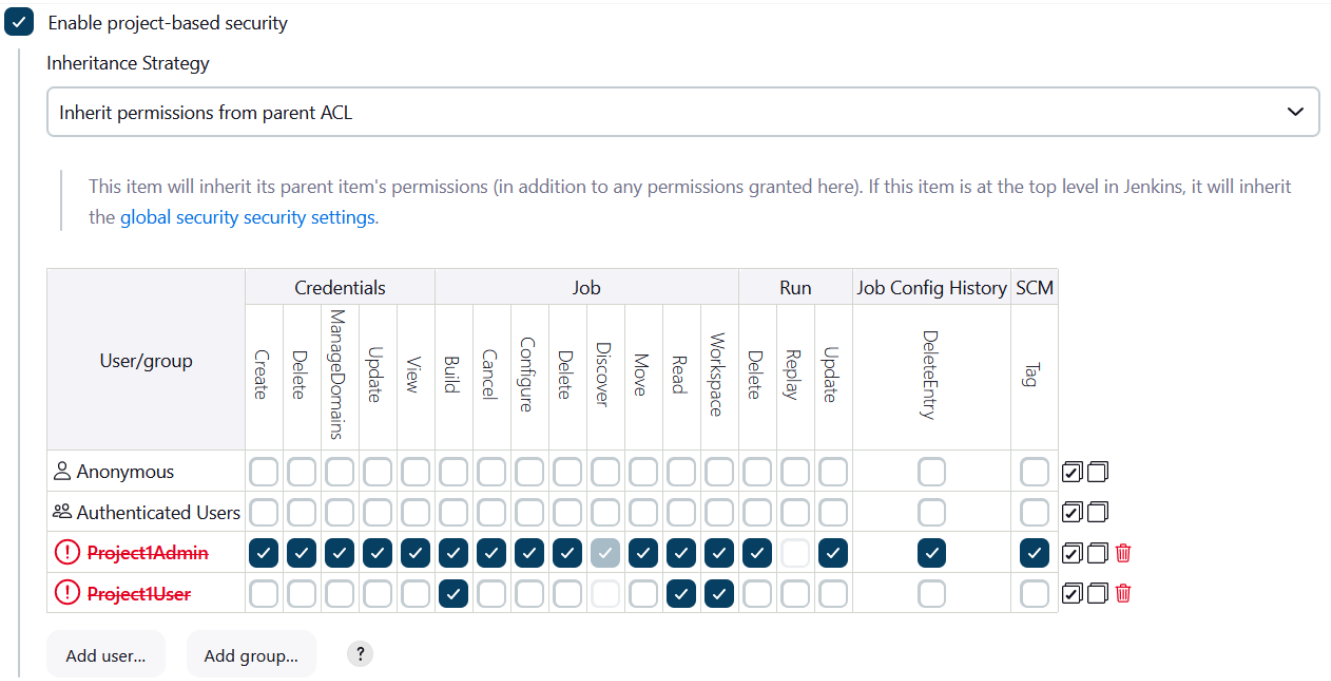
Access Rights Definition (job/project level)

Go to the "Configuration" page of a job dedicated to Project 1 and check "Enable project-based security":



Do the following changes in the table:

- Add a "Project1Admin" and give it all rights on this job by checking all check boxes,
- Add a "Project1User" and check "Read", "Build" and "Workspace" check boxes,



Do the same work on all jobs linked to Project1.

Repeat all above actions with "Project2Admin" and all jobs linked to Project2.

Result

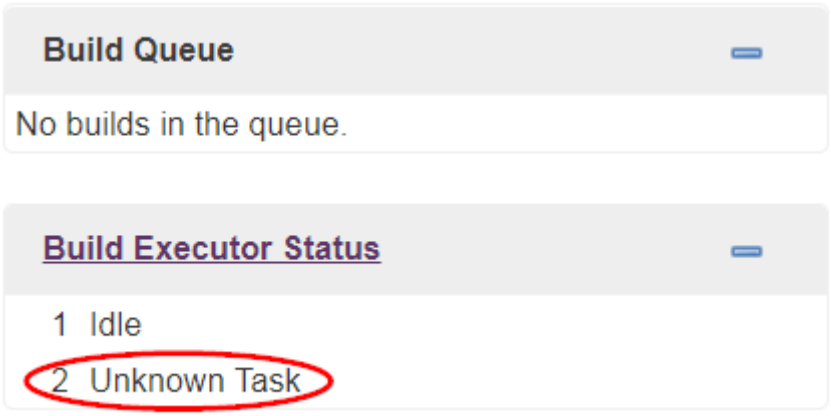
- SuperAdmin has full rights on the whole Jenkins instance,
- Project's admins see and have full rights on jobs linked to their projects (e.g. they can add new admins/users),
- Project's users can only see, launch, get logs and artifacts on jobs linked to their projects.

Known Limitations

Inter-project Information Sharing

An admin/user dedicated to a project will not be allowed to see information on jobs of other projects.

For example, when logged as Project2Admin and with Project1's server running.Project2Admin will see:



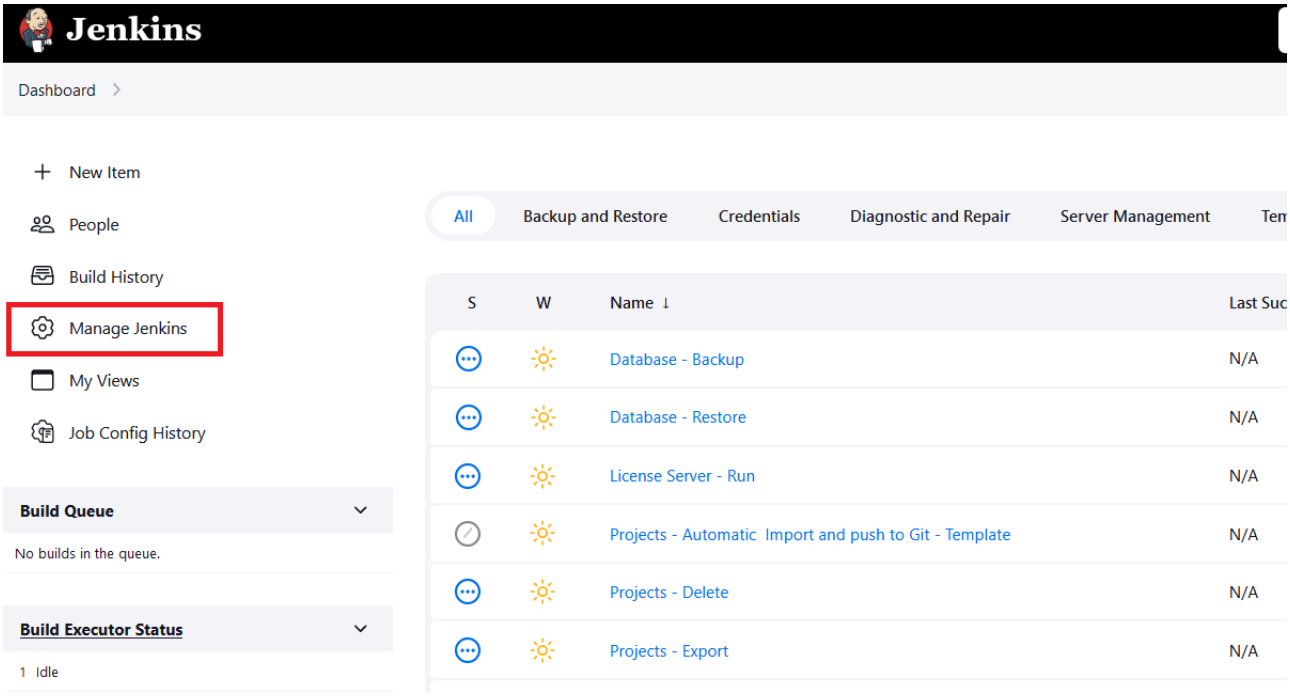
Tips and Tricks

Configure Number of Scheduler Build Processes

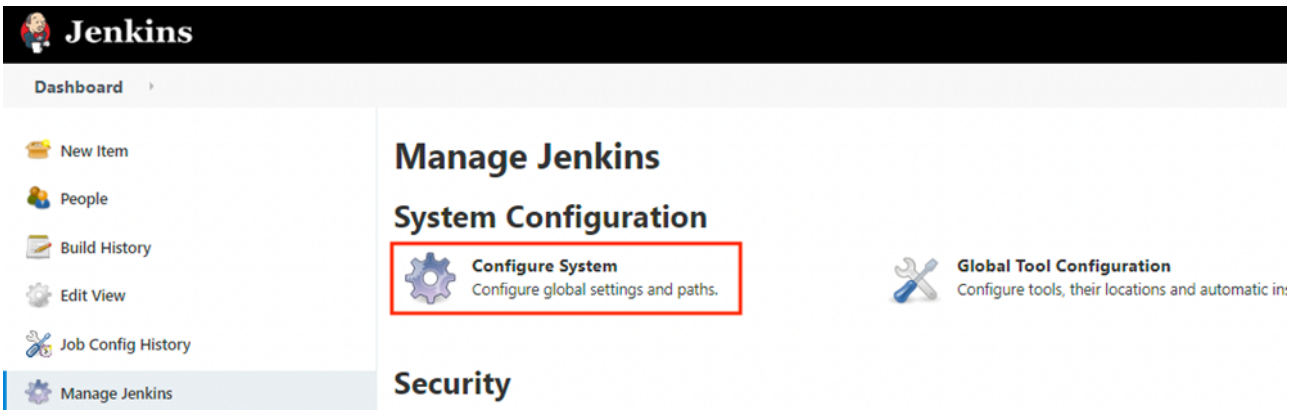
The Team for Capella scheduler (Jenkins) can be configured for a maximum number of build processes that can execute concurrently.

In order to ensure the correct operation of all Team for Capella server jobs it is vital to set this maximum number of build processes correctly!

1. Select ***Manage Jenkins***.

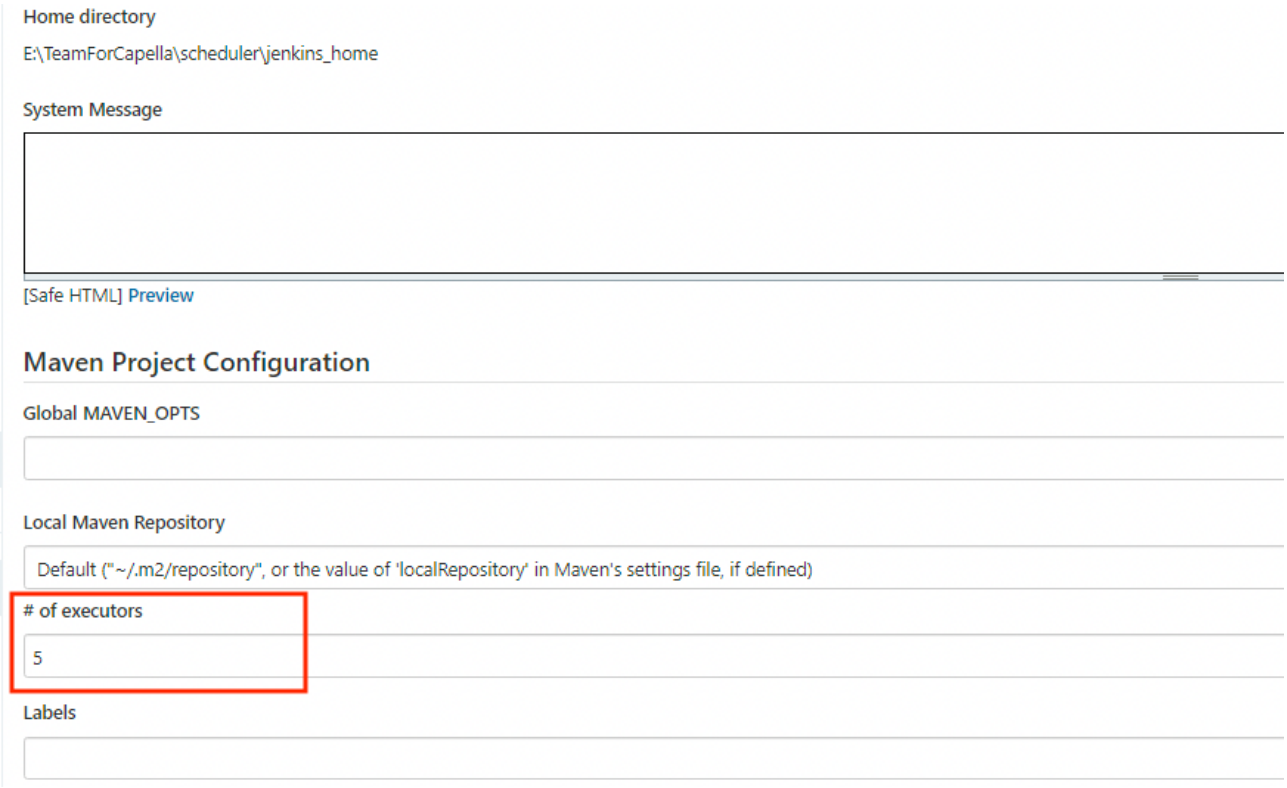


2. Select ***Configure System***.



3. Locate the setting ***# of executors*** and set the value according to the following rule:

```
# of executors = <total number TFC server processes to execute> + 1
```



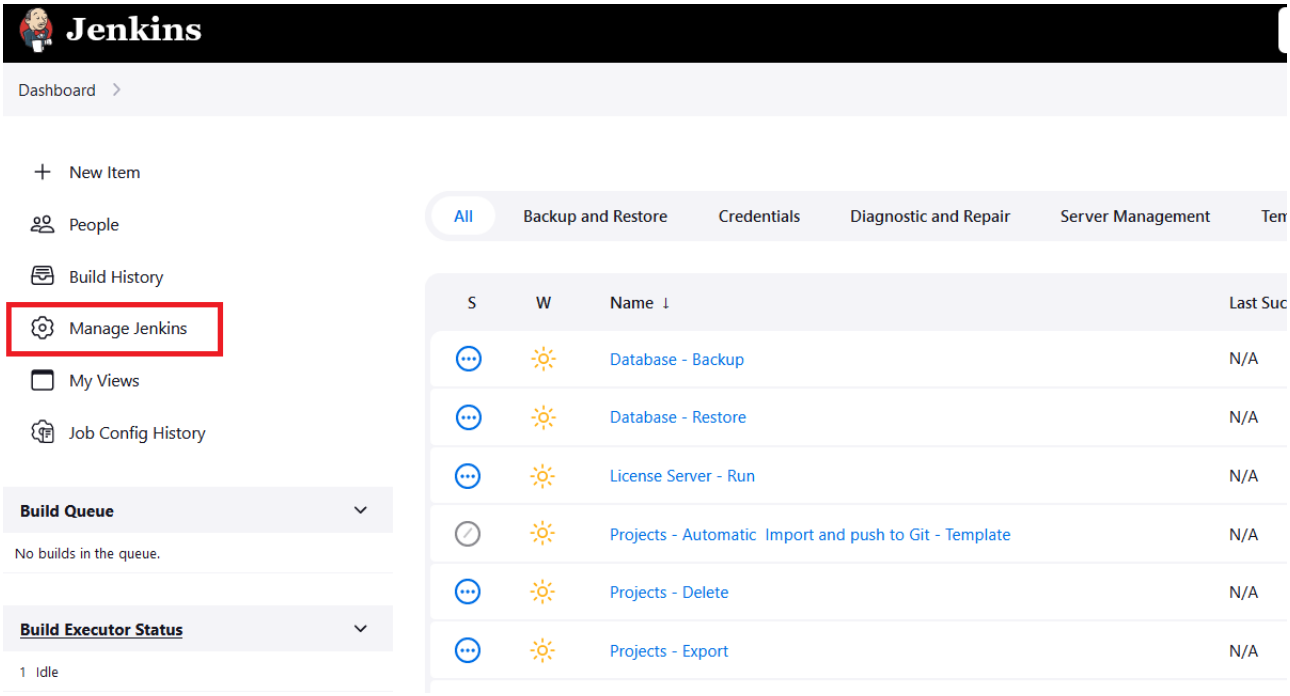
For example, if the server machine is to run 5 Team for Capella server processes, then the value of *# **of executors*** would need to be set to ***6***.

Setting this configuration parameter incorrectly can lead to complete system hangs, no Capella backups, etc!

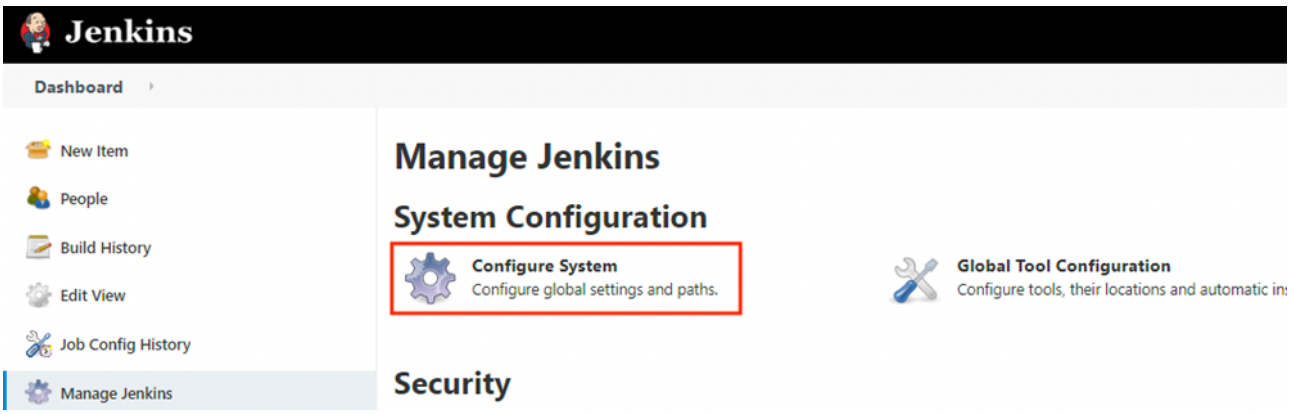
Create Scheduler Job Environment Variables

Each Team for Capella server process relies on two network ports – a server port and a console port. In order to avoid confusion by using "magic" numbers for the ports within the scheduler jobs, it is best to create environment variables for these.

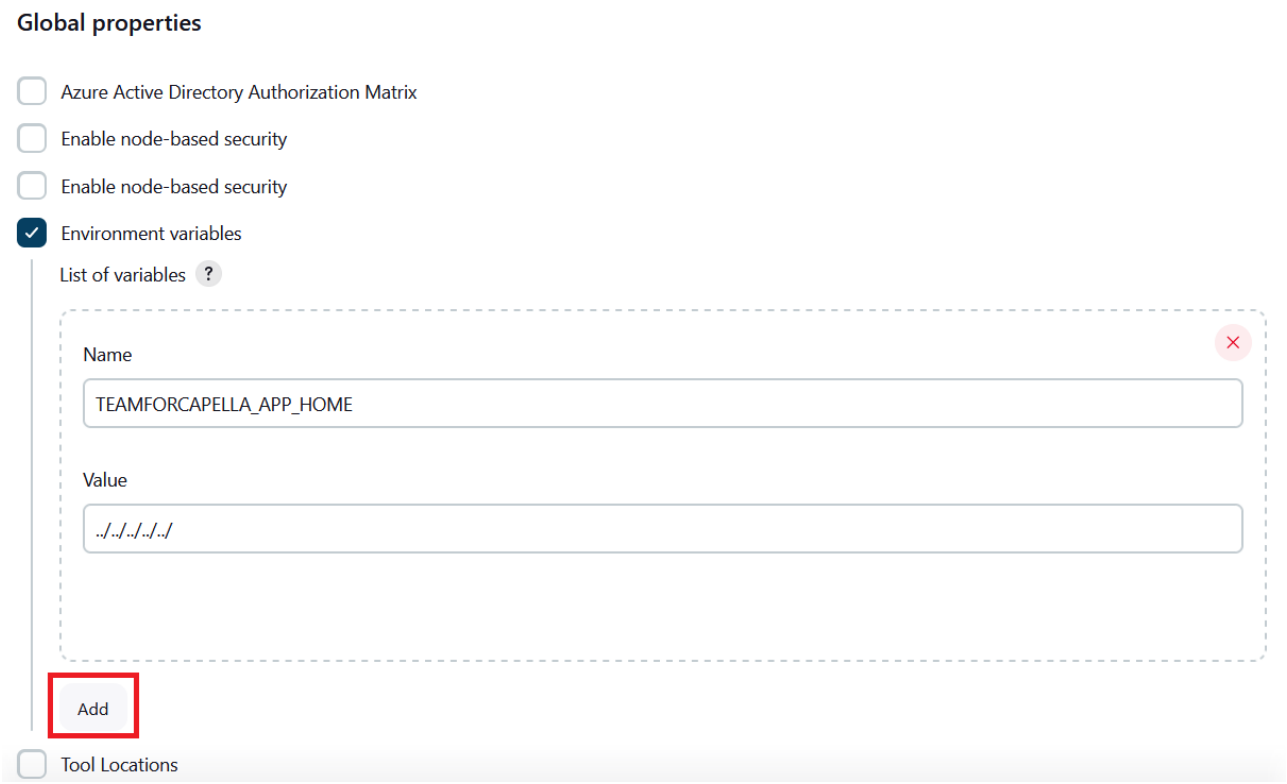
1. Select ***Manage Jenkins***.



2. Select ***Configure System***.



3. Within the section ***Global properties*** → ***Environment variables***, press the ***Add*** button in order to add a new variable.



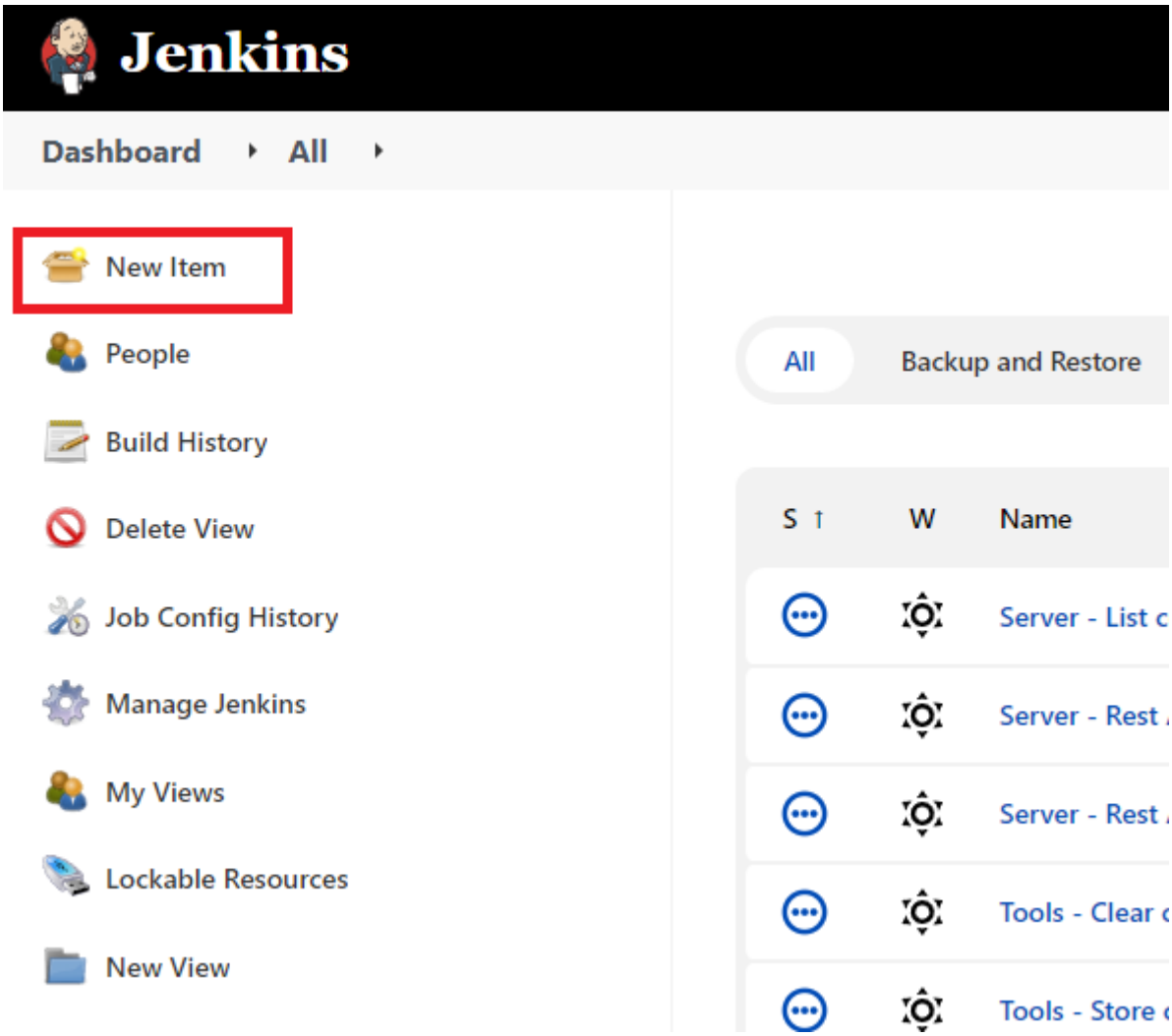
4. Enter the server port environment variable name and value as follows: Set ***name*** to ***TEAMFORCAPELLA_SERVER_PORT_<repoName>***, where ***<repoName>*** is replaced by the name of the repository, e.g., ***TEST_01*** Set ***value*** to the configured server port value, e.g., ***2036***.

5. Press the ***Add*** button in order to add a new variable.
6. Enter the console port environment variable name and value as follows: Set ***name*** to ***TEAMFORCAPELLA_CONSOLE_PORT_<repoName>***, where ***<repoName>*** is replaced by the name of the repository, e.g., ***TEST_01*** Set ***value*** to the configured console port value, e.g., ***12036***.

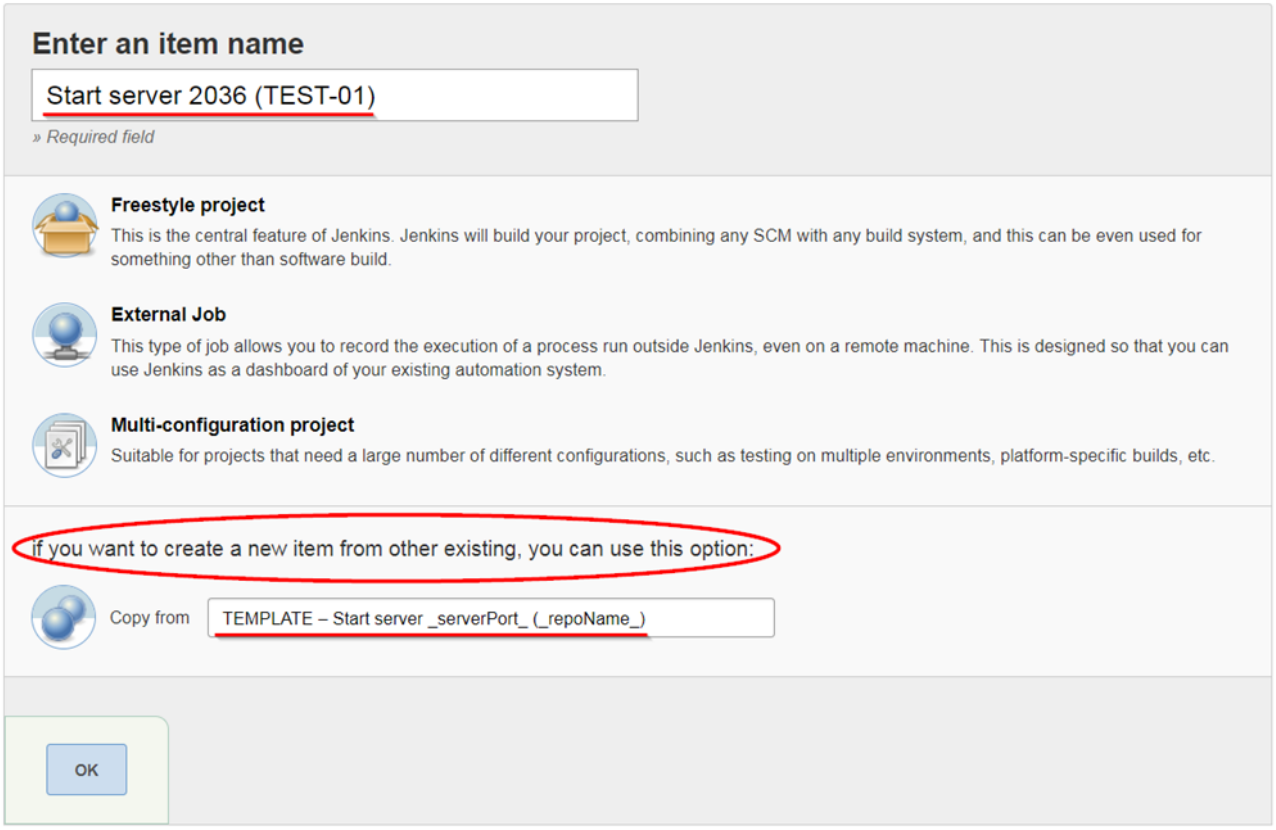
Note: the hyphen character is not allowed within the names of environment variables. Therefore, in the above example, although the repository name is test-01, within the environment variable name, the hyphen is replaced by an underscore, i.e., Test_01

Create an additional Server – Start Job

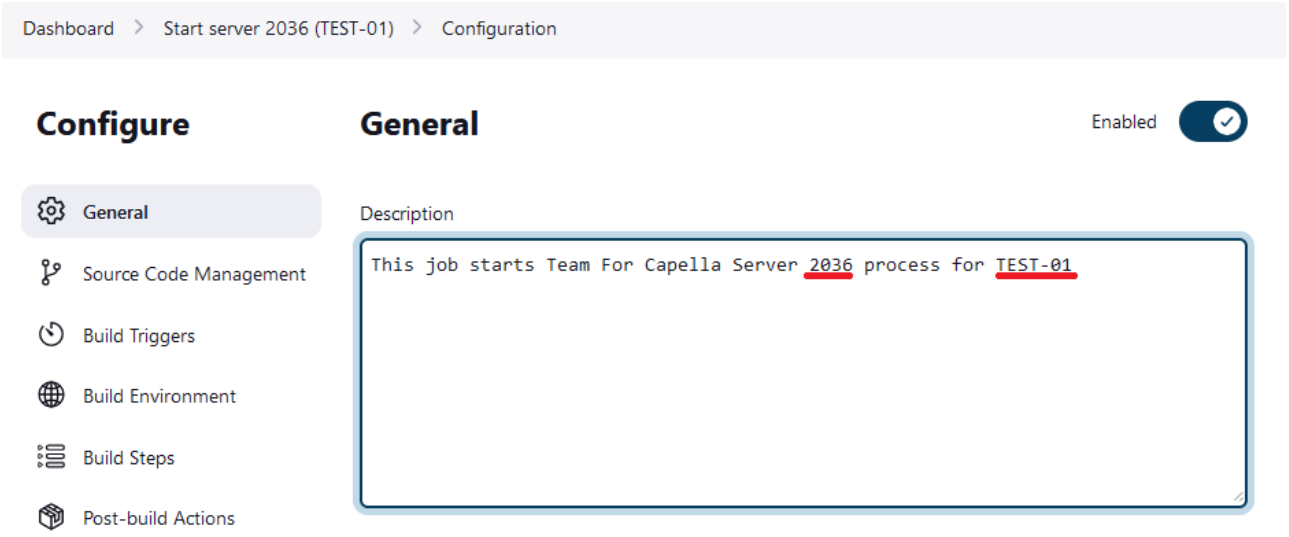
1. From the main page of the Team for Capella scheduler, select the ***New Item*** link from the menu on the left-hand side of the screen.



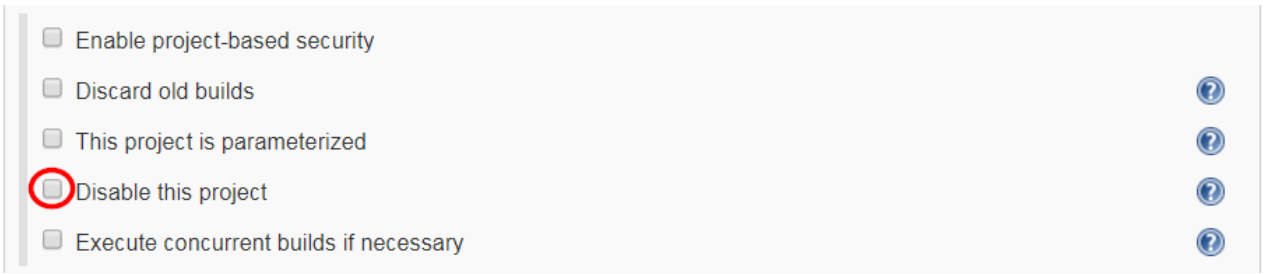
2. Enter the job name and source job template as follows: Set the ***Job name*** to "***Start server <serverPort> (<repoName>)***", where ***<serverPort>*** is replaced by the configured server port number, e.g. ***2036*** and ***<repoName>*** is replaced by the repository name, e.g. ***TEST-01***. Activate the ***Copy existing job*** radio button. In the ***Copy from*** text field, start typing the word "*TEMPLATE*" and then from the drop-down list that appears, select the entry "***TEMPLATE – Start server <serverPort> (<repoName>)***". Press ***OK***.



3. In the job configuration screen, amend the ***Description*** text by replacing the placeholders <***serverPort***> and <***repoName***> with the actual server port and repository name respectively.

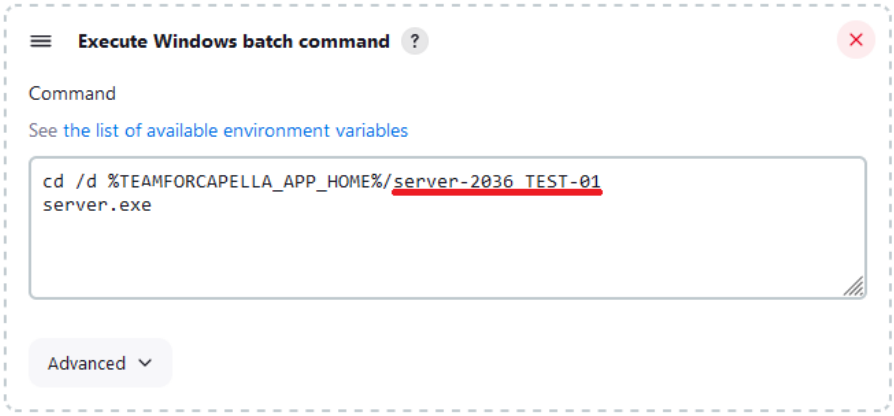


4. Activate the job by deselecting the ***Disable this project*** checkbox.

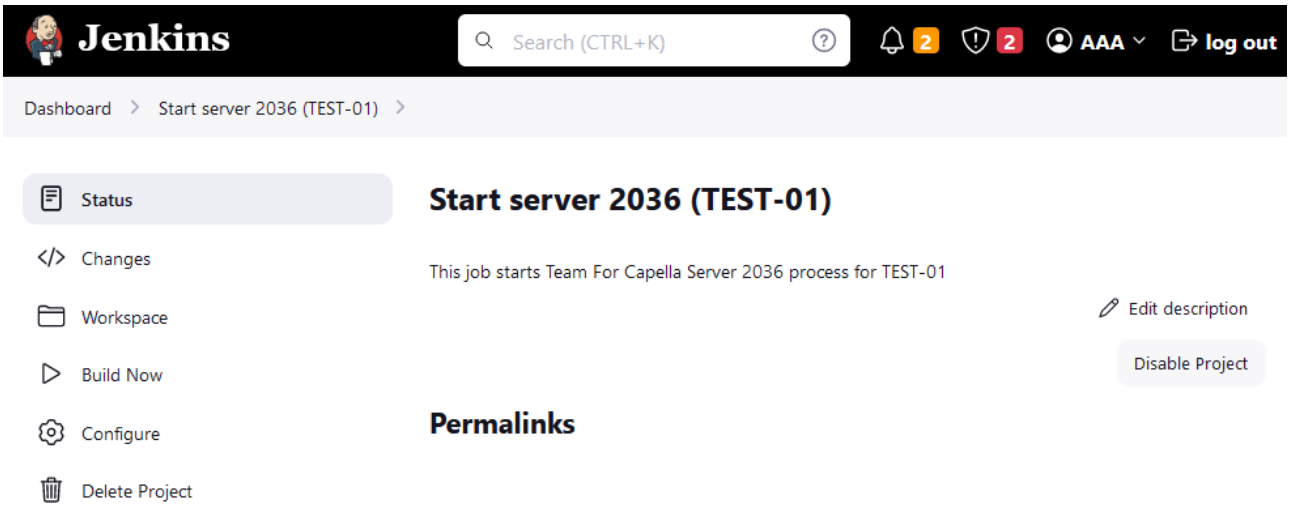


5. Modify the Team for Capella server path within the ***Command*** field of the ***Build*** section, replacing ***serverPort*** and ***repoName*** within the path name with the configured server port and

repository name respectively, for example:

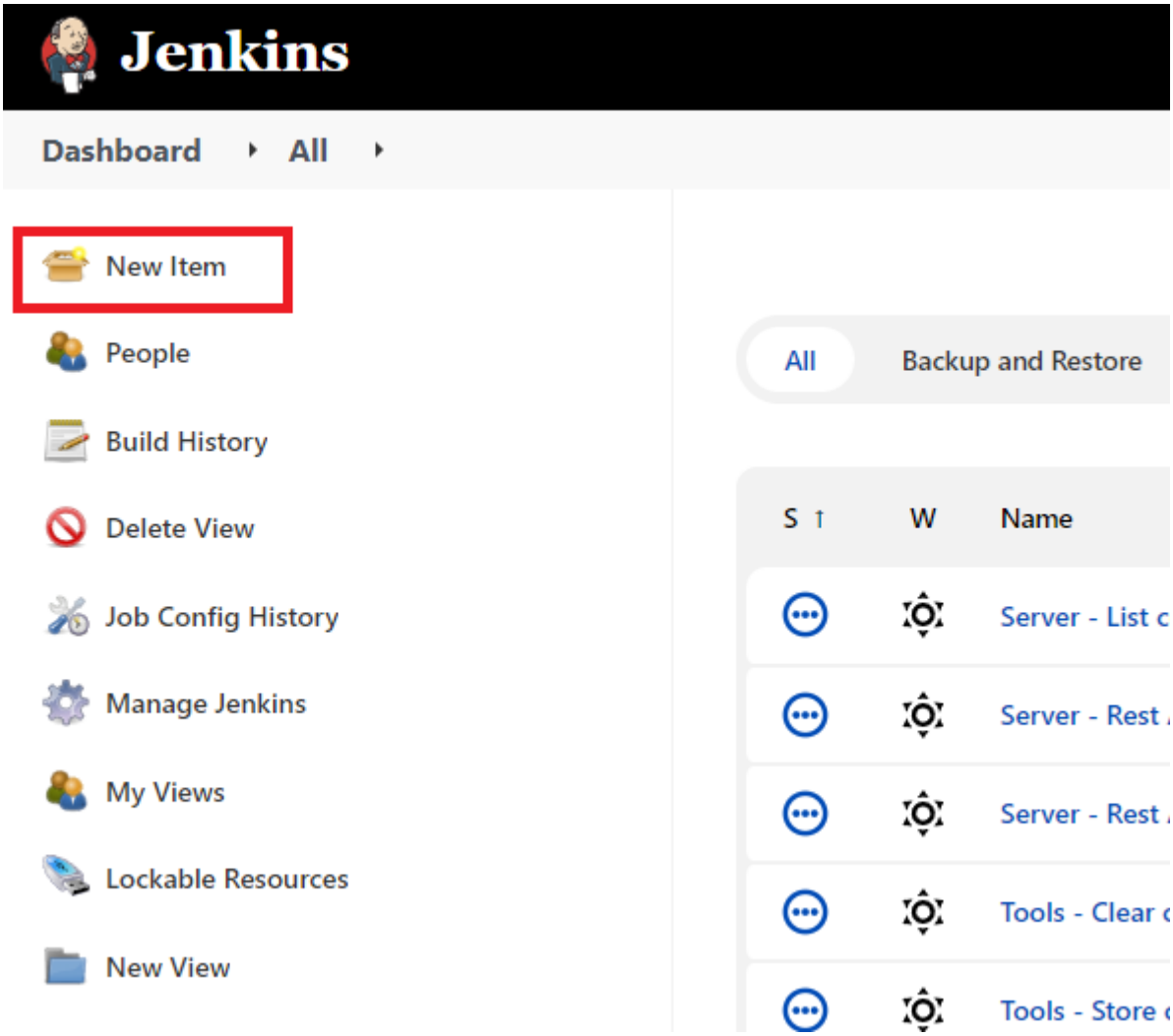


6. Upon saving the changes to the job the main screen for the new job appears.

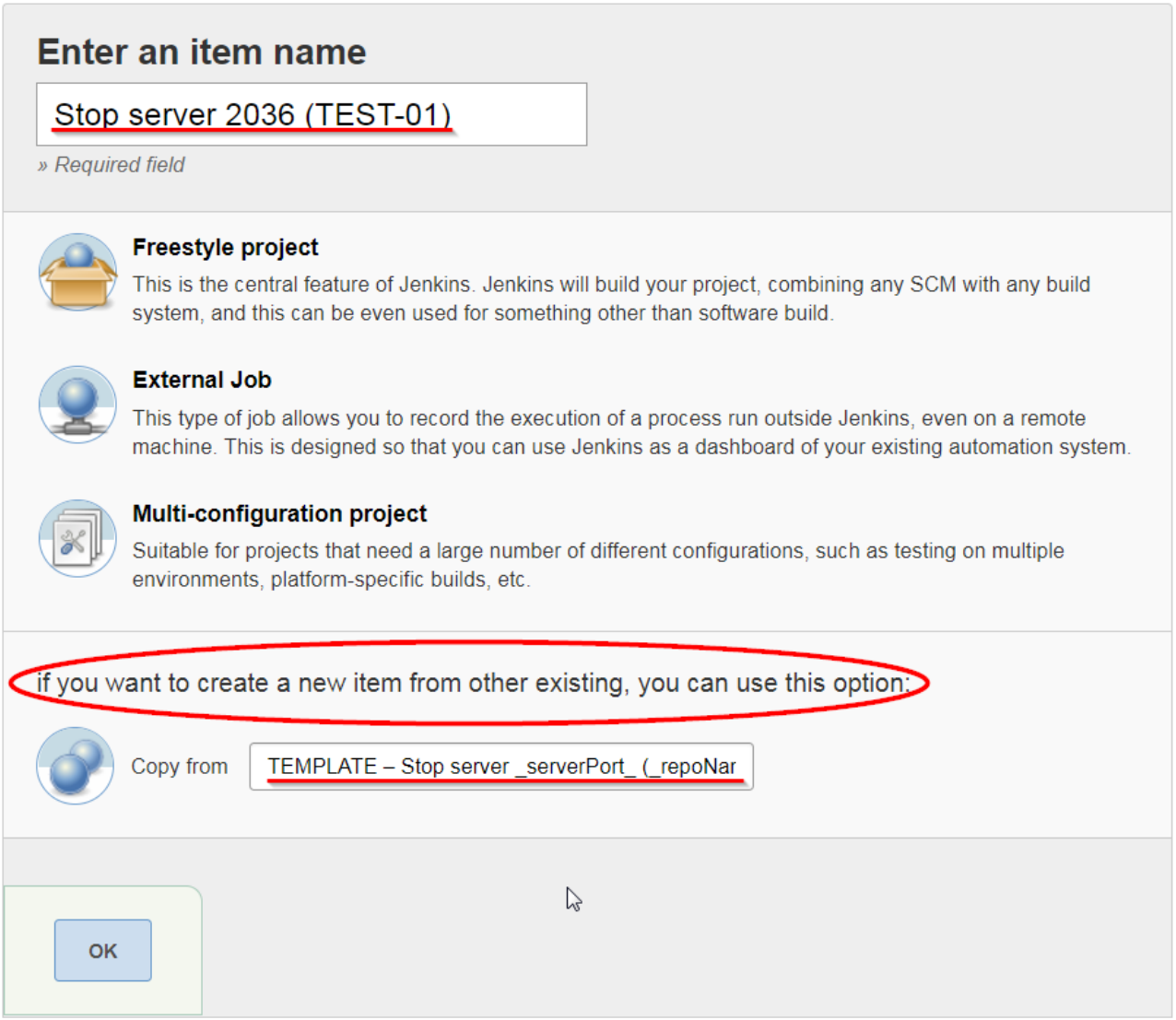


Create an additional Server – Stop Job

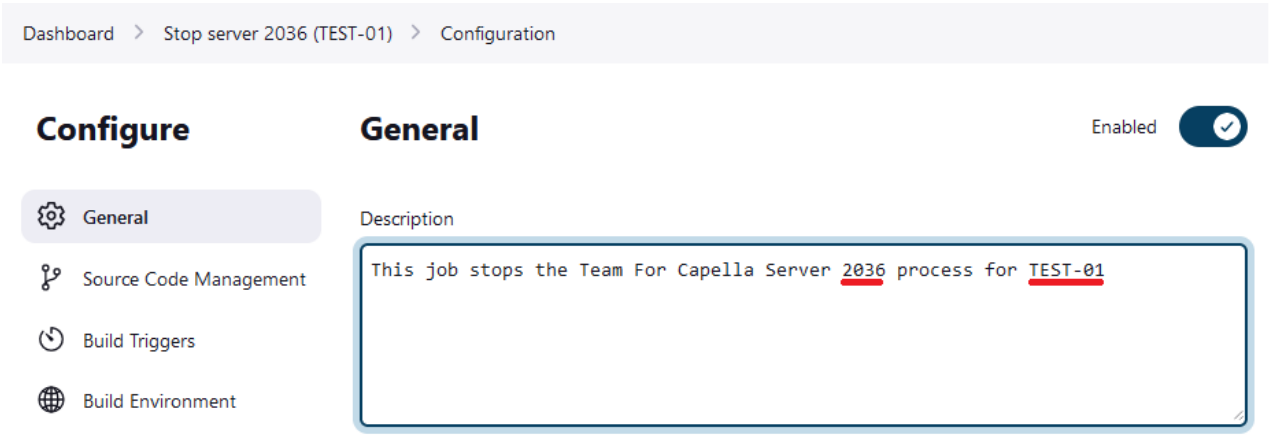
1. From the main page of the Team for Capella scheduler, select the ***New Item*** link from the menu on the left-hand side of the screen.



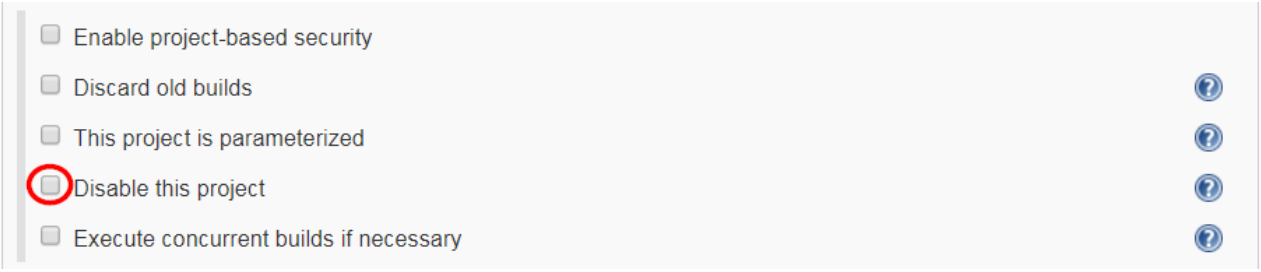
2. Enter the job name and source job template as follows: Set the ***Job name*** to "***Server – Stop <serverPort> (<repoName>)***", where ***<serverPort>*** is replaced by the configured server port number, e.g. ***2036*** and ***<repoName>*** is replaced by the repository name, e.g. ***TEST-01***. Activate the ***Copy existing job*** radio button. In the ***Copy from*** text field, start typing the word "*TEMPLATE*" and then from the drop-down list that appears, select the entry "***TEMPLATE – Stop server <serverPort> (<repoName>)***". Press ***OK***.



3. In the job configuration screen, amend the ***Description*** text by replacing the placeholders <***serverPort***> and <***repoName***> with the actual server port and repository name respectively.



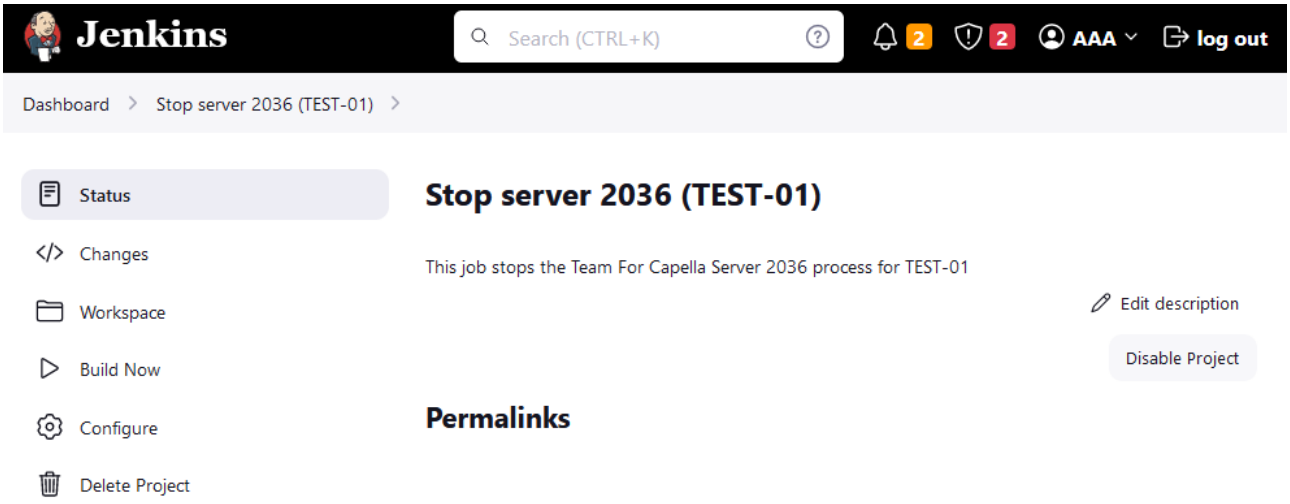
4. Activate the job by deselecting the ***Disable this project*** checkbox.



5. Modify the Team for Capella console port environment variable within the ***Command*** field of the ***Build*** section, replacing ***TEAMFORCAPELLA_CONSOLE_PORT_repoName*** with the appropriate console port environment variable for this Team for Capella server/repo, for example:

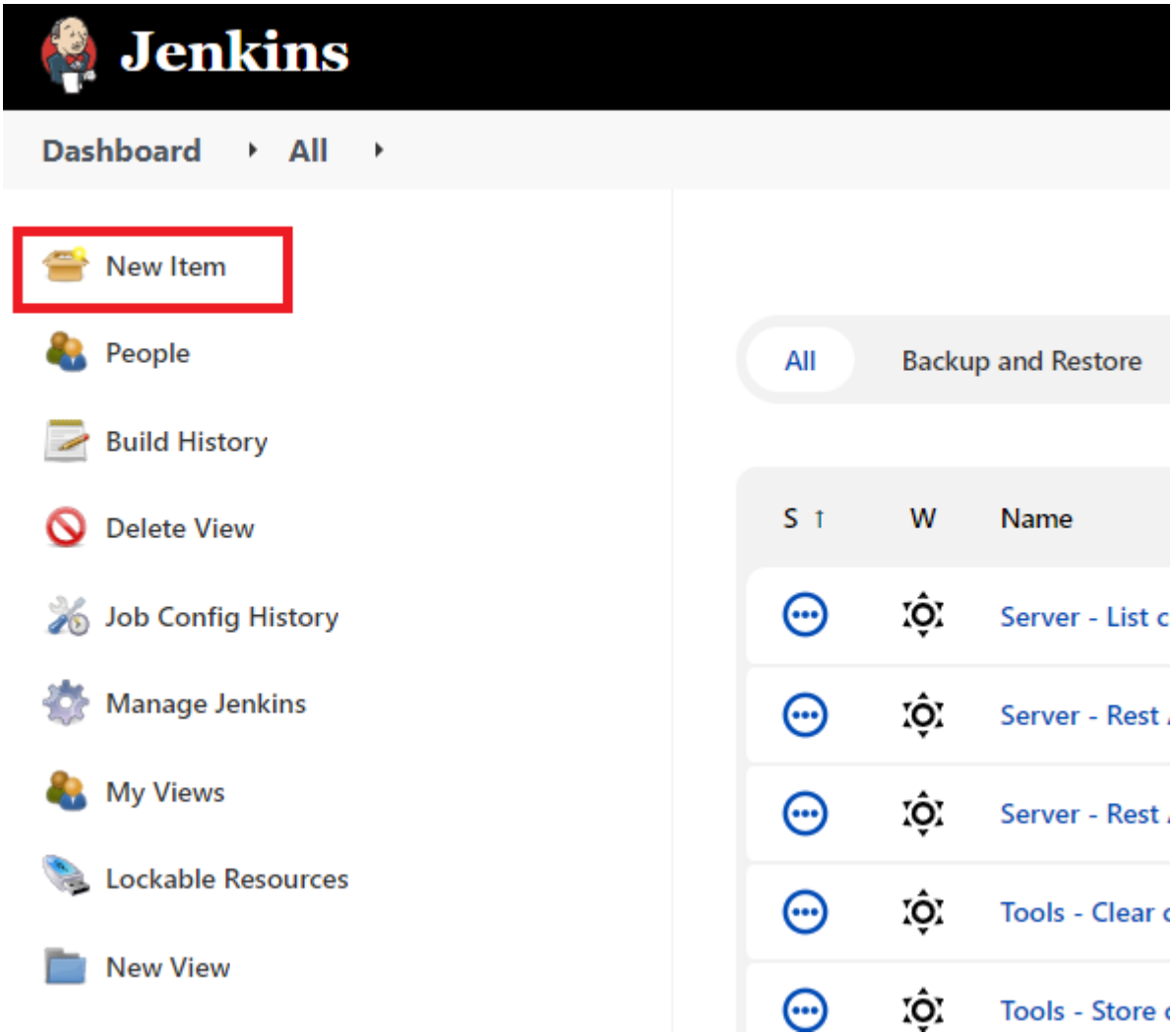
```
cd TEAMFORCAPELLA_APP_HOME/tools
command.bat -consoleLog localhost TEAMFORCAPELLA_CONSOLE_PORT_TEST_01 cdo
stopserver
```

6. Upon saving the changes to the job, the main screen for the new job appears.

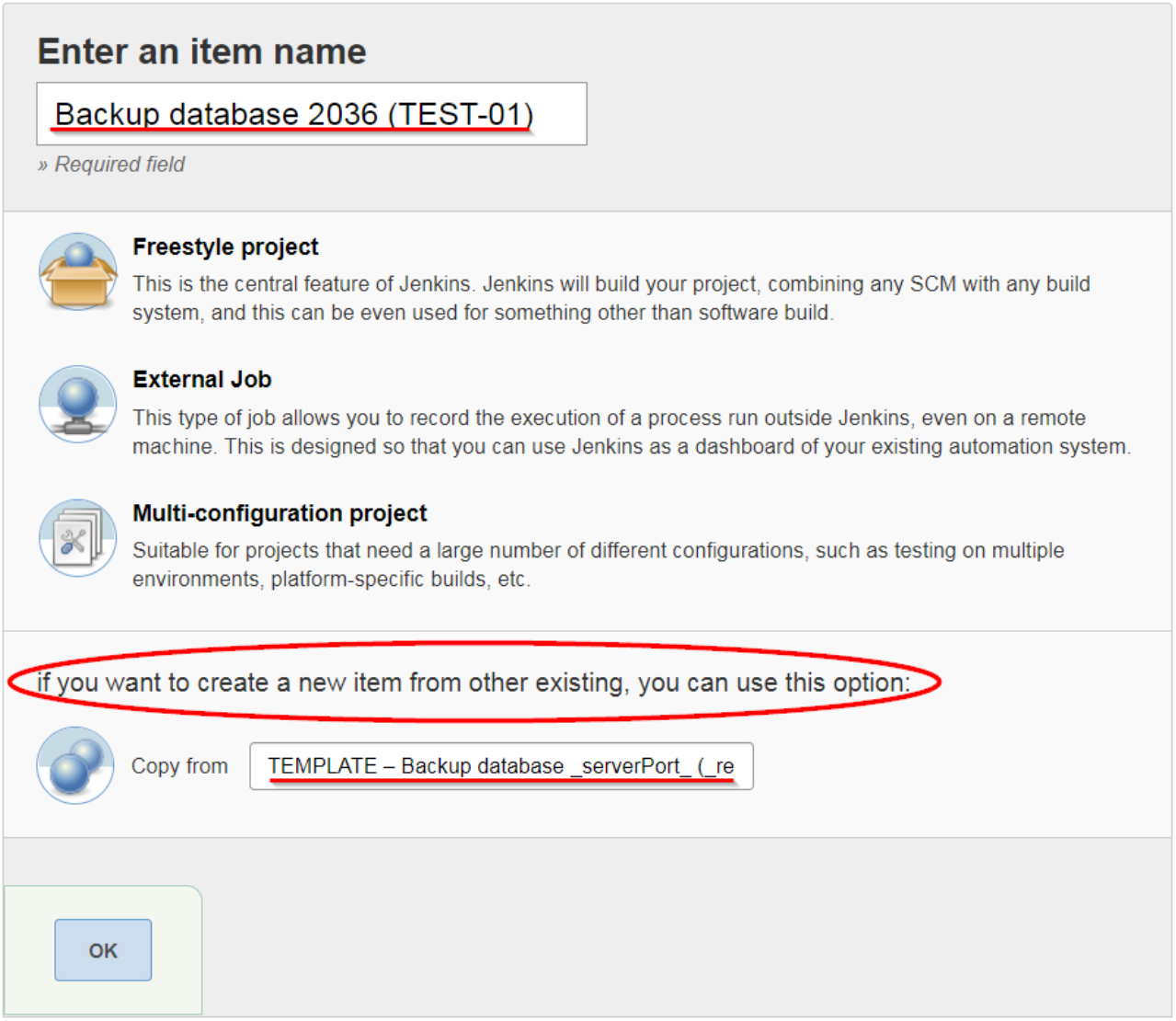


Create an additional Database – Backup Job

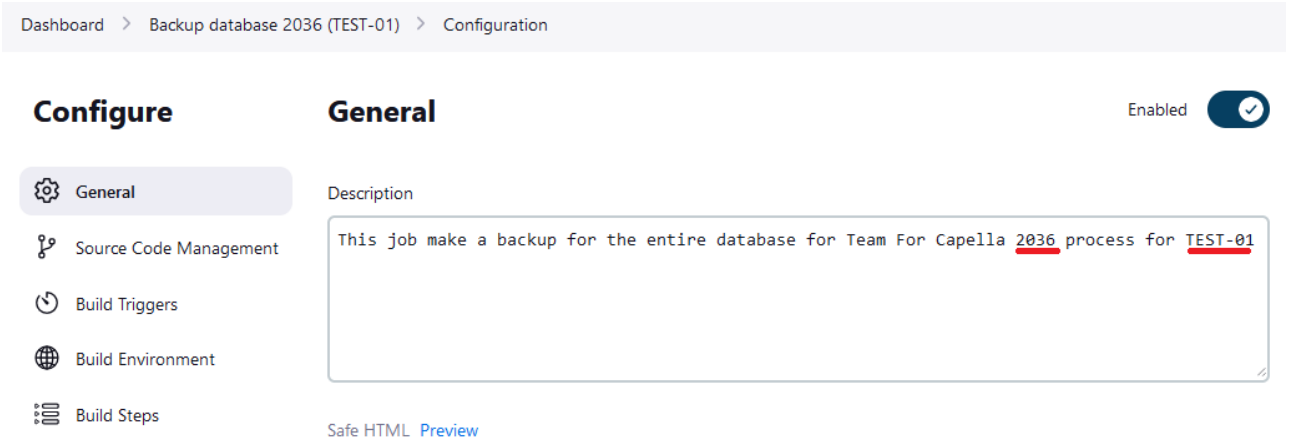
1. From the main page of the Team for Capella scheduler, select the ***New Item*** link from the menu on the left-hand side of the screen.



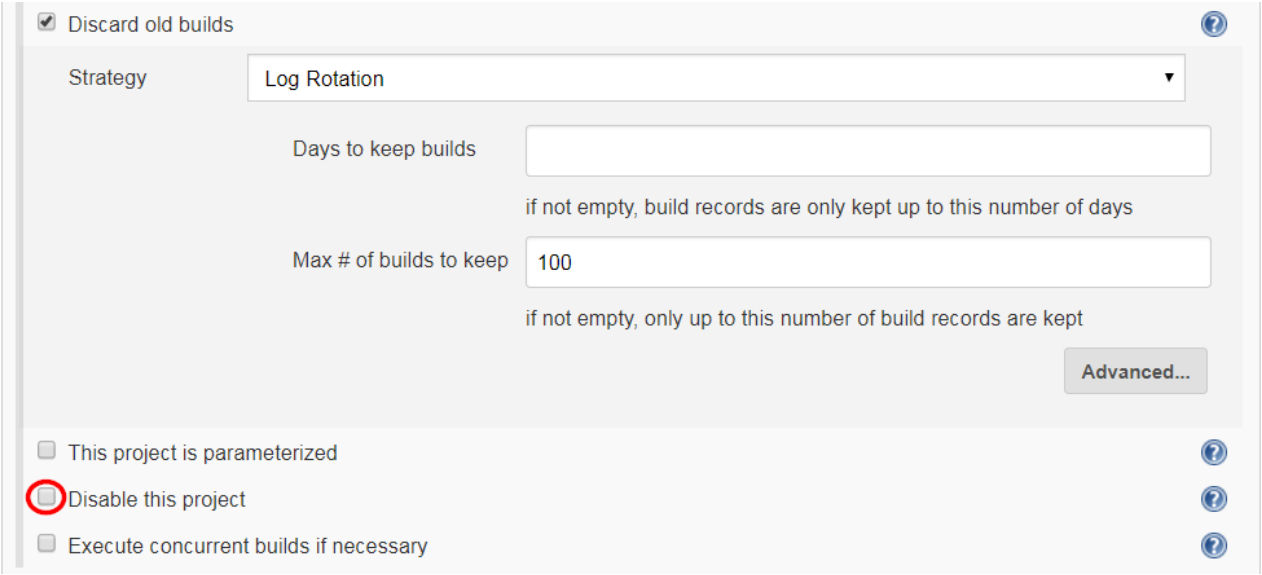
2. Enter the job name and source job template as follows: Set the ***Job name*** to "***Database – Backup <serverPort> (<repoName>)***", where ***<serverPort>*** is replaced by the configured server port number, e.g., ***2036*** and ***<repoName>*** is replaced by the repository name, e.g., ***TEST-01***. Activate the ***Copy existing job*** radio button. In the ***Copy from*** text field, start typing the word *TEMPLATE*" and then from the drop-down list that appears, select the entry "***TEMPLATE – Database – Backup <serverPort> (<repoName>)***". Press ***OK***.



3. In the job configuration screen, amend the ***Description*** text by replacing the placeholders <***serverPort***> and <***repoName***> with the actual server port and repository name respectively.



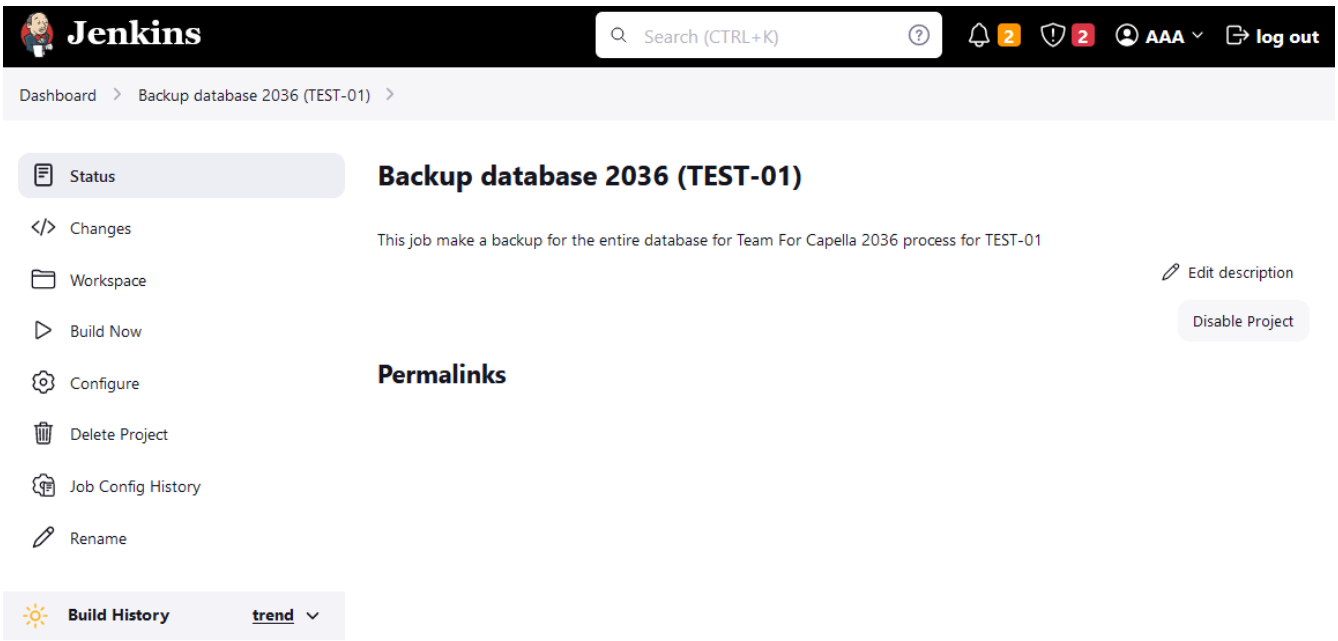
4. Activate the job by deselecting the ***Disable this project*** checkbox.



5. Modify the Team for Capella console port environment variable within the ***Command*** field of the ***Build*** section, replacing ***TEAMFORCAPELLA_CONSOLE_PORT_repoName*** with the appropriate console port environment variable for this Team for Capella server/repo, for example:

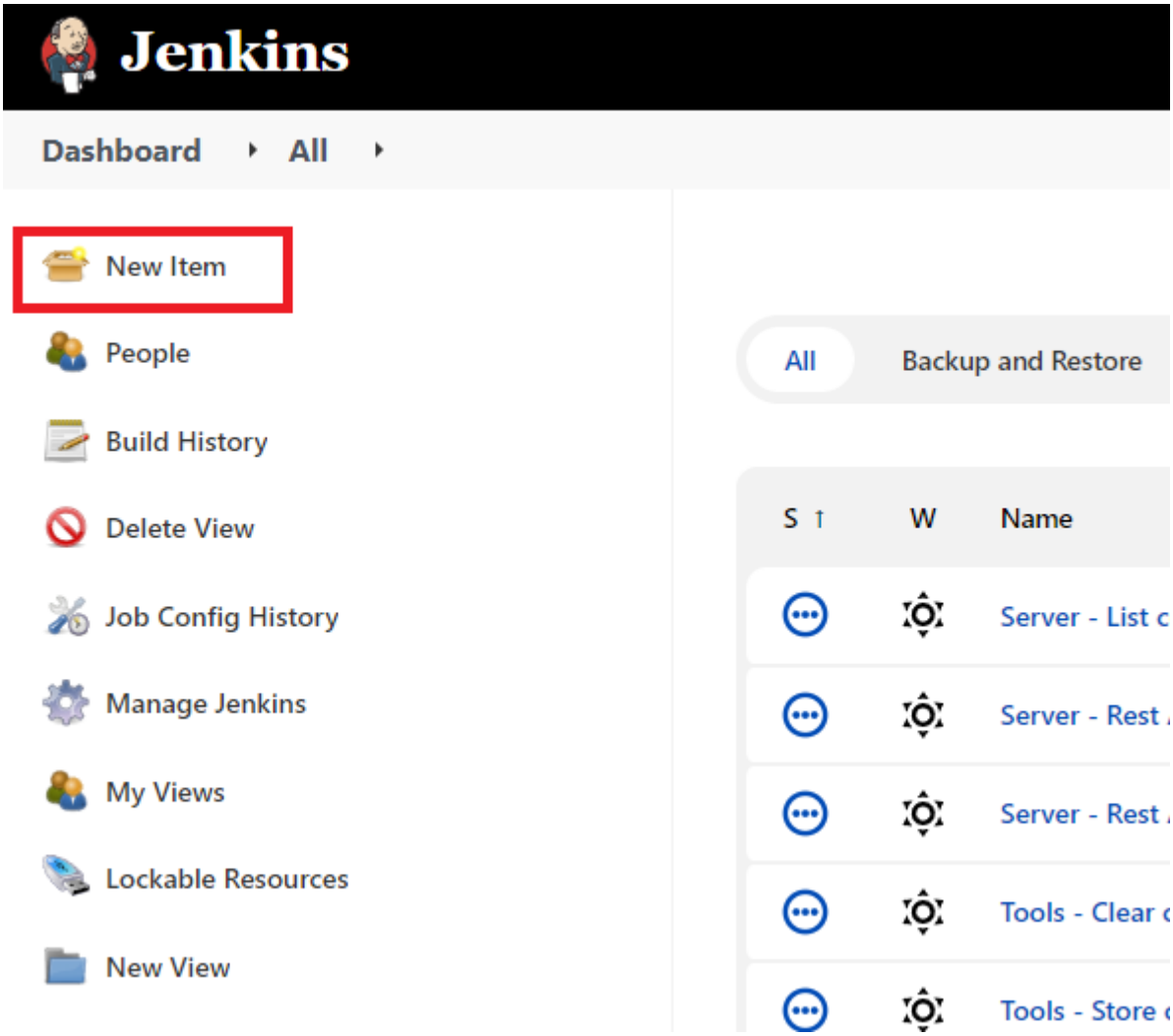
```
del *-sql.zip
cd TEAMFORCAPELLA_APP_HOME/tools command.bat -consoleLog localhost
TEAMFORCAPELLA_CONSOLE_PORT_TEST_01 capella_db backup ' WORKSPACE'
```

6. Upon saving the changes to the job the main screen for the new job appears.

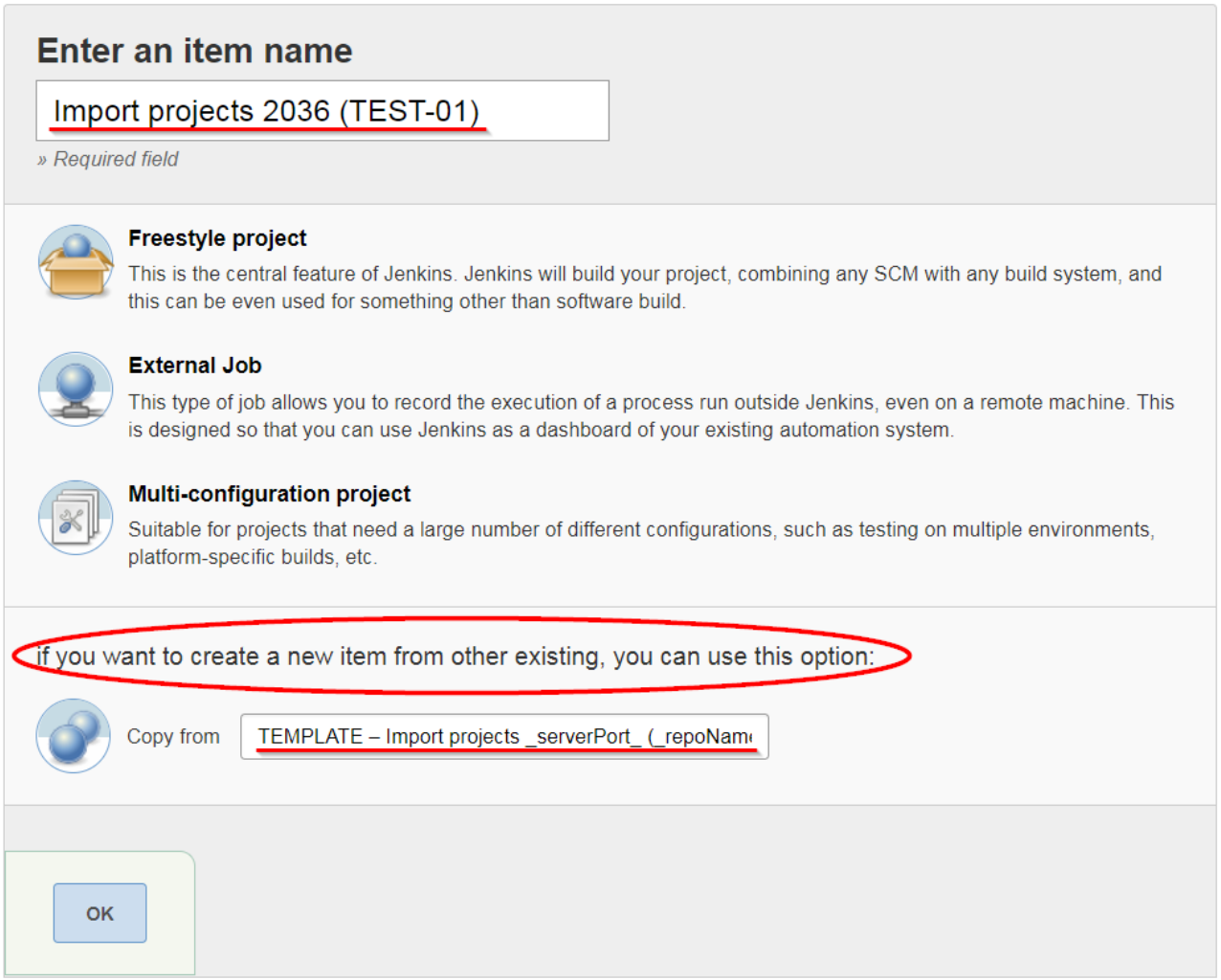


Create an additional Projects – Import Job

1. From the main page of the Team for Capella scheduler, select the ***New Item*** link from the menu on the left-hand side of the screen.

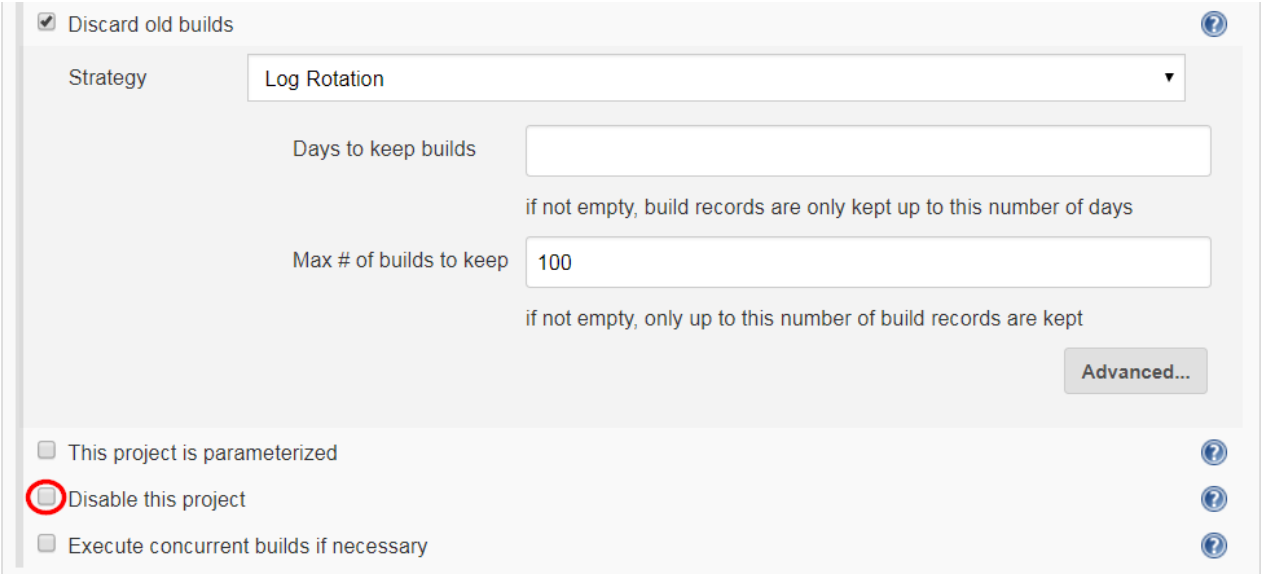


2. Enter the job name and source job template as follows: Set the ***Job name*** to "***Projects – Import <serverPort> (<repoName>)***", where ***<serverPort>*** is replaced by the configured server port number, e.g., ***2036*** and ***<repoName>*** is replaced by the repository name, e.g., ***TEST-01***. Activate the ***Copy existing job*** radio button. In the ***Copy from*** text field, start typing the word "*TEMPLATE*" and then from the drop-down list that appears, select the entry "**TEMPLATE – Projects – Import <serverPort> (<repoName>)**". Press ***OK***.

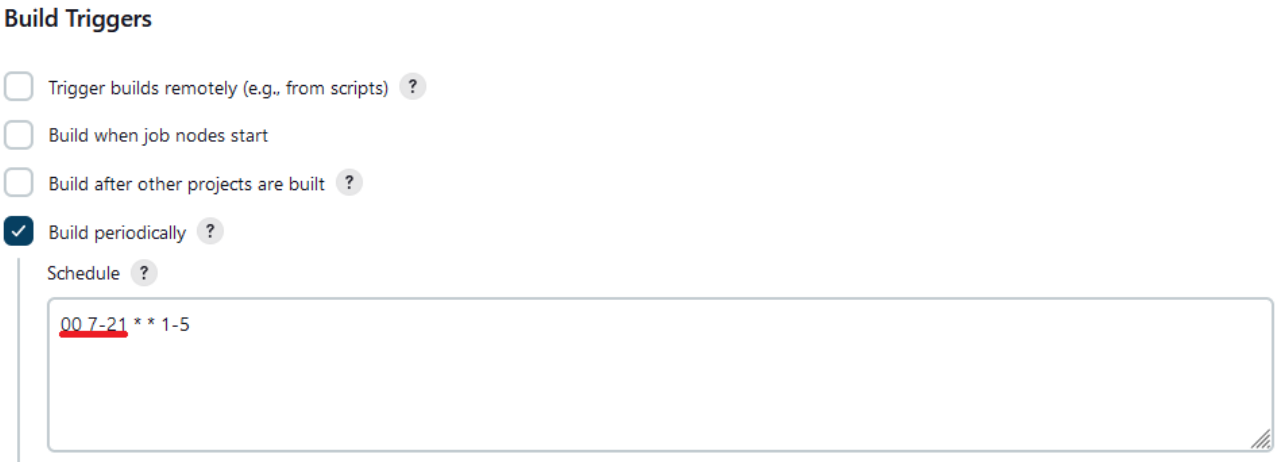


3. In the job configuration screen, amend the ***Description*** text by replacing the placeholders <***serverPort***> and <***repoName***> with the actual server port and repository name respectively.

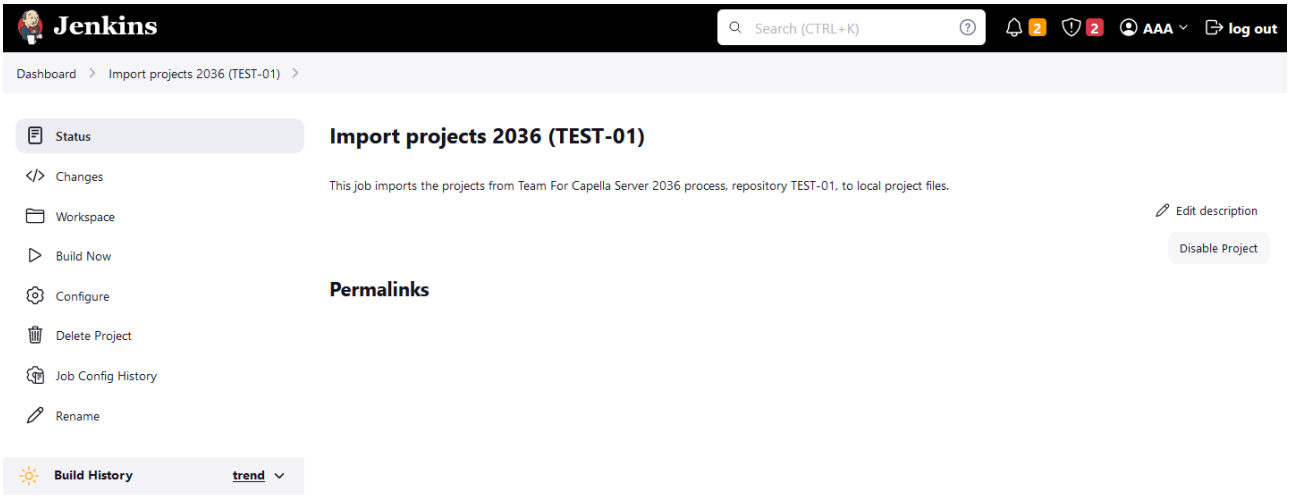
4. Activate the job by deselecting the ***Disable this project*** checkbox.



5. It is not recommended to have multiple Import jobs launched at the same time. Each Import job must be shifted in time by at least 30 minutes. In the job configuration, in ***Build Triggers*** section, modify the ***minutes*** and ***hours*** values within the schedule (first and second numeric *cron* fields) if needed.



6. Within the ***Command*** field of the ***Build*** section, modify the Team for Capella server and console ports environment variables and Team for Capella repository name as follows: Replace ***TEAMFORCAPELLA_SERVER_PORT_repoName*** with the appropriate server port environment variable for this Team for Capella server/repo Replace ***TEAMFORCAPELLA_HTTP_PORT*** with the appropriate rest API server port environment variable for this Team for Capella server/repo Replace <***repoName***> with the name for this Team for Capella repository:bc. del *.zip del *.txt del *.activitymetadata rd /s /q importer-workspace cd TEAMFORCAPELLA_APP_HOME/tools importer.bat -data "WORKSPACE/importer-workspace" -outputFolder "WORKSPACE" -archiveProject true -stopRepositoryOnFailure true -checksize 5 -importCommitHistoryAsText -port TEAMFORCAPELLA_SERVER_PORT_repoName -httpPort TEAMFORCAPELLA_HTTP_PORT -repoName TEST_01
7. Upon saving the changes to the job, the main screen for the new job appears.



Troubleshooting

Jenkins window service is not launched when there are multiple versions of Java installed

By default, Jenkins will be launched using the java executable found in Windows\System. If the java version from this java executable is different from the key Java Runtime Environment\CurrentVersion in the registry, the service cannot be installed. If this problem is encountered, there are 2 solutions:

- Make sure that the version of the key Java Runtime Environment\CurrentVersion is the same as the java executable found in Windows\System.
- Modify the jenkins.xml to replace java executable by the absolute path to the chosen installed java.

4.3. Importer Configuration

The importer is an application used to extract the project from the cdo server database to a local folder. It produces as many zip files as modeling projects. It can also be used to import the user profiles model.

The importer also extracts information from the CDO Commit history to produce a representation of the activity made on the repository. This information is denominated *Activity metadata*. See help chapter The commit history view and Commit description preferences for a complete explanation. By default, the importer will extract *Activity Metadata* for every commit in the repository. Be aware that the parameter *-projectName* has no impact on this feature. It will also export commits that do not impact the selected project. Still, it is possible to specify a range of commits using the parameters *-to* and *-from*.

Importer strategies

Several import strategies are supported by the Importer application:

- *Connected import*: the Importer application establishes a connection to the targeted repository and imports the models.
  - This is the default strategy of the Importer application.

    - Credentials might be required if the server has been configured to use identification, authentication or user profiles, see Server Configuration job documentation.

- *Offline import*: This mode allows performing the import based on a snapshot of the targeted repository.
    - There is no connection to the server and no interaction with other users: no credentials are required for the Importer application.
    - It also avoids overloading the server and can be done in a separate environment.
    - It can be enabled with the use of the `-importFilePath` parameter. Refer to this parameter documentation in the next section for more details.
    - A snapshot is an XML extraction of the repository. It can be manually obtained by executing the `cdo export` command on the server osgi console.
- *Snapshot import*: the Importer application sends a snapshot creation command to the server, then it uses the created snapshot to perform an *Offline Import*.
    - This is the strategy used by the *Project - Import* Scheduler job.
    - Since the XML extraction is more efficient than the *Connected import*, this option keeps most of the benefit of the simple *Offline import*.
    - It can be enabled with the use of the `-cdoExport true` parameter alongside with `-importFilePath` which define where to create and then consume the snapshot.
    - **Note:** With this strategy, a **lock preventing any commit** from connected users is acquired. During the time of the snapshot execution, it is not possible for connected users to commit their changes. The lock is released once the snapshot is over. If the lock cannot be acquired (after three attempts), the import is abandoned. The attempt number (three by default) can be overridden through a system property. For instance, to replace the number of attempts by two: `-Dcom.thalesgroup.mde.melody.importer.maxAttemptsCdoExport=2`

See also Projects — Import job documentation.

Importer parameters

Importer.bat file uses **-vmargs** as a standard eclipse parameter. Eclipse parameters that are used by *importer.bat* override the value defined in *capella.ini* file. So if you want to change a system property existing in *capella.ini* (-vmargs -Xmx3000m for example) do not forget to do the same change in *importer.bat*.

The importer needs credentials to connect to the CDO server if the server has been started with authentication or user profile. Credentials can be provided using either `-repositoryCredentials` or `-repositoryLogin` and `-repositoryPassword` parameters. Credentials are required only for *Connected import* (see the *Importer strategies* section above for more details). Here is a list of arguments that can be set to the Importer (in importer.bat or in a launch config):

Arguments	Description
-repositoryCredentials	Login and password can be provided using a credentials file. It is the recommended way for confidentiality reason. If the credentials file does not contain any password, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage This parameter must not be used with `-repositoryLogin` or `-repositoryPassword` parameters else the importer will fail. To use this property file • Add the following program argument: `-repositoryCredentials <path_to_credentials_file>` • Fill the specified file using the following format (only one line allowed): `aLogin:aPassword` Note: Credentials are required only for *Connected import* (see the *Importer strategies* section above for more details).
-repositoryLogin	The importer needs a login in order to connect to the CDO server if the server has been started with authentication or user profile. `-repositoryPassword` must not be used with `-repositoryCredentials` else the application will fail. Note: Credentials are required only for *Connected import* (see the *Importer strategies* section above for more details).
-repositoryPassword	This parameter is used to provide a password to the importer accordingly to the login. If `-repositoryPassword` is not used, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage `-repositoryPassword` must not be used with `-repositoryCredentials` else the application will fail. **Warning:** some special characters like double-quote might not be properly handled when passed in argument of the importer. The recommended way to provide credentials is through the repositoryCredentials file or the secure storage. Note: Credentials are required only for *Connected import* (see the *Importer strategies* section above for more details).

Arguments	Description
-hostname	Define the team server hostname (default: localhost).
-port	Define the team server port (default: 2036).
-connectionType	The connection kind can be set to **tcp** or **ssl** (keep it in low-case) (default: tcp)
-httpLogin	Importer application will trigger an Http request. This argument allows giving a login to identify with on the Jetty server.
-httpPassword	Importer application will trigger an Http request. This argument allows giving a password to authenticate with on the Jetty server.
-httpPort	Importer application will trigger an Http request. This argument allows giving a port to communicate with on the Jetty server.
-httpsConnection	Importer application will trigger an Http request. This boolean argument specifies if the connection should be Https or Http.
-importType	The backup is available in five different modes: • **PROJECT_ONLY** to only export the shared modeling projects from the CDO repository to local; • **SECURITY_ONLY** to only export the shared user profile project from the CDO repository to local; • **COMMIT_HISTORY_ONLY** to only export the shared user profile project from the CDO repository to local; • **PROJECT_AND_COMMIT_HISTORY** to only export the shared user profile project from the CDO repository to local; • **ALL** to export projects, security model and commit history. (default: PROJECT_AND_COMMIT_HISTORY)
-repoName	Define the team server repository name (default: repoCapella).
-projectName	By default, all projects are imported, i.e. with default value "*" (with the right -importType parameter). Argument "-projectName X" can be used to import only project X. Encoded and decoded names are both accepted (e.g. your can use either -projectName "Prj A", -projectName Prj%20A or -projectName "Prj%20A") (default: *).
-runEvery	Import every x minutes (default -1: disabled).

Arguments	Description
-outputFolder	Define the folder where to import projects (default: workspace).
-logFolder	Define the folder where to save logs (default: -outputFolder).
-archiveProject	Define if the project should be zipped (default: true). Each project will be zipped in a separate archived suffixed with the date. Some additional archives can also be created: • For projects containing images referenced by the current project: If the current project being managed by the importer process contains a diagram element that has a reference to an image which is located in another project, then this other project will be added in another zip file. See more information about image management • For Capella libraries: If the current project being managed by the importer process has a dependency to a library, then the resource of the library used by the current project will be part of another zip file. Note: Some library resources may not be referenced by the current project and so not included in the zip.
-overrideExistingProject	If the output folder already contains a project with the same name, this argument allows removing this existing project.
-closeServerOnFailure	Ask to close the server on project import failure (default: false). If the server hosts several repositories, it is better to use the parameter -stopRepositoryOnFailure.
-stopRepositoryOnFailure	Ask to stop the repository on project import failure (default: false). Note: it is currently not possible to restart a single repository, if defined in cdo-server.xml. To restart the stopped repository, stop and restart the server.
-backupDBOnFailure	Backup the server database on project import failure (default: true).
-checkSize	Check project zip file size in Ko under which the import of this project fails (default: -1(no check)).

Arguments	Description
-checkSession	Do some checks and log information about each imported project (default: true). • It checks that the project session can be opened and closed and that it contains no resource with an URI with the scheme cdo. • It also logs a lot of useful information about the project: used viewpoints, information about representations and capella models. For more details, refer to Sirius Session Details of the Sirius user documentation.
-errorOnInvalidCDOUri	Raise an error on cdo uri consistency check (default: true).
-addTimestampToResultFile	Add a time stamp to result files name (.zip, logs, commit history) (default: true).
-optimizedImportPolicy	This option is no longer available since 1.1.2.
-maxRefreshAttemptBeforeFailure	The max number of refresh attempt before failing (default: 10). If the number of attempts is reached, the import of a project will fail, but as this is due to the activity of remote users on the model, this specific failure will not close the repository or the server even with "-stopRepositoryOnFailure" or "-closeserveronfailure" set to true.
-timeout	Session timeout used in ms (default: 60000).
-importImages	Define which images should be imported. Possible values are 'ALL', 'USED', 'NONE'. (default : ALL).
-checkout	The timestamp specifying the date corresponding to the state of the projects that will be imported (refer to the following note for details on the accepted formats). If empty, the framework will connect on the repository with a transaction on the repository head (current state). When this parameter is used, the framework will open a read-only view on the repository at the given time instead of a transaction on repository head. This option is meaningful only if -importType is one of ALL, PROJECT_ONLY, SECURITY_ONLY or PROJECT_AND_COMMIT_HISTORY.

Arguments	Description
-from	The timestamp specifying the date from when the metadata will be imported (refer to the following note for details on the accepted formats). If omitted, it imports from the first commit of the repository. The timestamp can also be computed from an '‹Activity Metadata›' model. In that case, this parameter could either be an URL or a path in the file system to the location of the model. If the date corresponds to a commit, this commit is included. Otherwise the framework selects the closest commit following this date. In the case of using a previous activity metadata, the last commit of the previous import is also included.
-to	The timestamp specifying the latest commit used to imported metadata (refer to the following note for details on the accepted formats). If omitted, it imports to the last commit of the repository. The framework selects the closest commit preceding this date. **''Be careful: this parameter only impacts the range of commit for importing activity metadata from the CDO server. Using this parameter will not import the version of the model defined by the given date. See -checkout argument for that purpose.**''
-squashCommitHistory	Squash consecutive commits done by the same user with the same description (default: true).
-importCommitHistoryAsText	Import commit history in a text file using a textual syntax (default: false). The file has the same path as the commit history model file, but with **txt** as extension.
-importCommitHistoryAsJson	Import commit history in a json file format (default: false). The file has the same path as the commit history model file, but with **json** as extension.
-includeCommitHistoryChanges	Import the commit history detailed changes for each commit done by a user with one of the save actions (default: false). The changes of commits done by wizards, actions and command line tools are not computed, those commits have a description which begins by specific tags like [Export], [Delete], [Maintenance], [User Profile], [Import], [Dump]. This option is applied for all kinds of commit history exports (xmi, text or json files). Warning about the importer performance: if this parameter is set to true, the importer might take more time, particularly if the history of commits is long.

Arguments	Description
-computeImpactedRepresentationsForCommitHistoryChanges	Compute the impacted representations while exporting changes (default: false). Warning about the importer performance: if this parameter is set to true, the importer might take more time, particularly if the history of commits is long. For each commit with changes to export, it will compute the impacted representations.
-importFilePath	This option allows performing the import based on an XML or binary extraction of the repository. It is mandatory for *Offline* and *Snapshot* imports, see the *Importer strategies* section for more details. It is recommended to provide an absolute path. A path ending with ".bin" will trigger the binary load, other extensions will trigger the XML load. Some arguments related to the server connection will be ignored. Only the arguments -outputFolder and -repoName are mandatory.
-XMLImportFilePath	(deprecated) see -importFilePath
-cdoExport	This option allows sending a snapshot creation command to the server before performing the import as described in the *Importer strategies* section. (default: false). The `-importFilePath` argument is mandatory since the path is used to create and consume the snapshot. A path ending with ".bin" will trigger the binary export, other extensions will trigger the XML expor. **Note:** The cdo export command takes the lock on project aird resources. This strategy makes it possible to prevent a concurrency save from connected users. If the lock cannot be acquired after several attempts, an error message is logged and the import is canceled.
-archiveCdoExportResult	This option defines if the XML file resulting from the cdo export command launched by the importer in intermediate step (if -cdoExport is true) should be zipped (default : false). If the option is true, the XML file zip is created in the "Output folder" (see -outputFolder documentation) and the XML file is then deleted. `-archiveCdoExportResult` must not be used without `-cdoExport` argument to true otherwise the application will fail. Indeed, the application will only archive the XML file if it has produced it.

Arguments	Description
-eclipse.keyring `<file_path>`	This option specifies the location of the secure storage file used by Eclipse to store password and sensitive informations. The secure storage helps protect confidential data by encrypting it with a master key. By default, the secure storage file is located in the *.eclipse* folder of the user directory. You can change this location by using the `-eclipse.keyring` option with the desired file path. For example, if you want to use the file `C:\Users\Alice\secure_storage`, you can launch Eclipse with the option `-eclipse.keyring C:\Users\Alice\secure_storage`.
-eclipse.password `<file_path>`	This option specifies the file path containing the master key password used by Eclipse to encrypt data in the secure storage. By default, the secure storage system uses a password provider mechanism to protect the master password used to encrypt data in the secure storage. If the `-eclipse.password` option is used, the password provider mechanism will not be used, a custom password file is used as master password. For example, if you want to use the file `C:\Users\Alice\password.txt` as master password for your application, you can launch Eclipse with the option `-eclipse.password C:\Users\Alice\password.txt`. This option can be used with `-eclipse.keyring` if you want to use a specific secure storage and a specific master password.
-help	Print help message.

The timestamp arguments (checkout, from, to) must use the following formats:

- yyyy-MM-ddThh:mm:ss.SSSZ
- The timezone may be omitted (format without Z part). In this case, the time zone is the time zone of the system.
- the time since epoch in milliseconds,
- the value "HEAD" for the latest available commit.

For example, for the date 03/08/2017 10h14m28s453ms on a time zone +0100 use the argument "2017-08-03T10:14:28.453+0100".

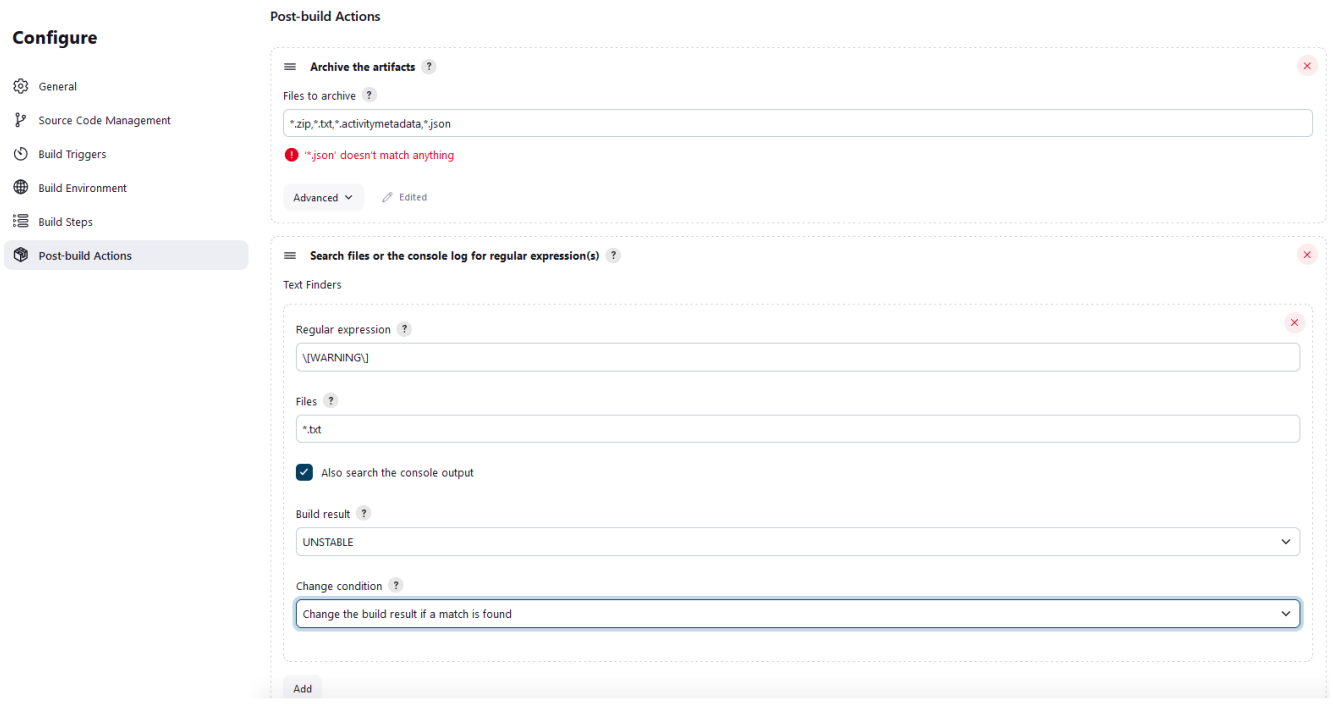
If the server has been started with user profile, the Importer needs to have write access to the whole repository (including the user profiles model). See Resource permission pattern examples section.

If this recommendation is not followed, the Importer might not be able to correctly prepare the model (proxies and dangling references cleaning, ...). This may lead to a failed import.

The importer uses the default configuration of Capella and does not need its own configuration area. For this to work properly, the importer needs to have read/write permission to the configuration area of Capella, otherwise it can end up with some errors about access being denied. A common situation where the importer can be found in this situation is when the Scheduler is launched as a Windows service. In this case, the user account executing the service is not necessarily configured to have the read/write permission to Capella's configuration area. If somehow you cannot give the read/write permission to the importer, a workaround is to provide it a dedicated configuration area by adding the following arguments at the end of importer.bat file: **-Dosgi.configuration.area="path/to/importer/configuration/area"** and if necessary, update the existing argument **-data importer-workspace** to point to a location with read/write permission.

Jenkins Text Finder configuration

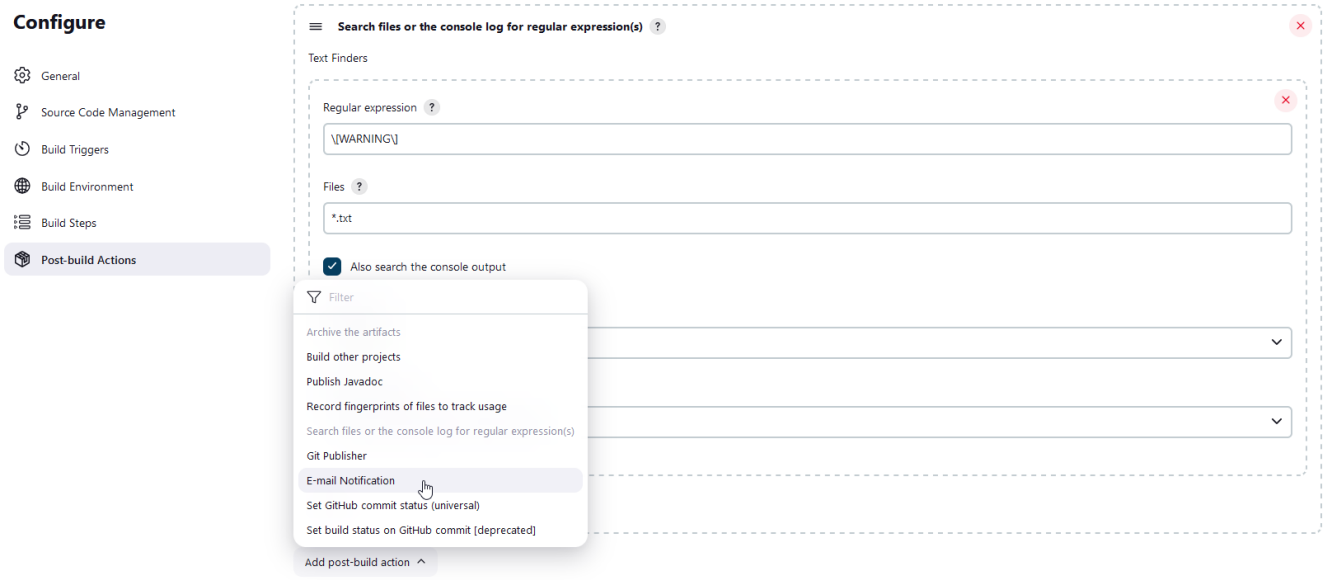
The job contains a post-action that verifies that the commit History metadata text file is generated with the parameter `exportCommitHistory` set to true by default:



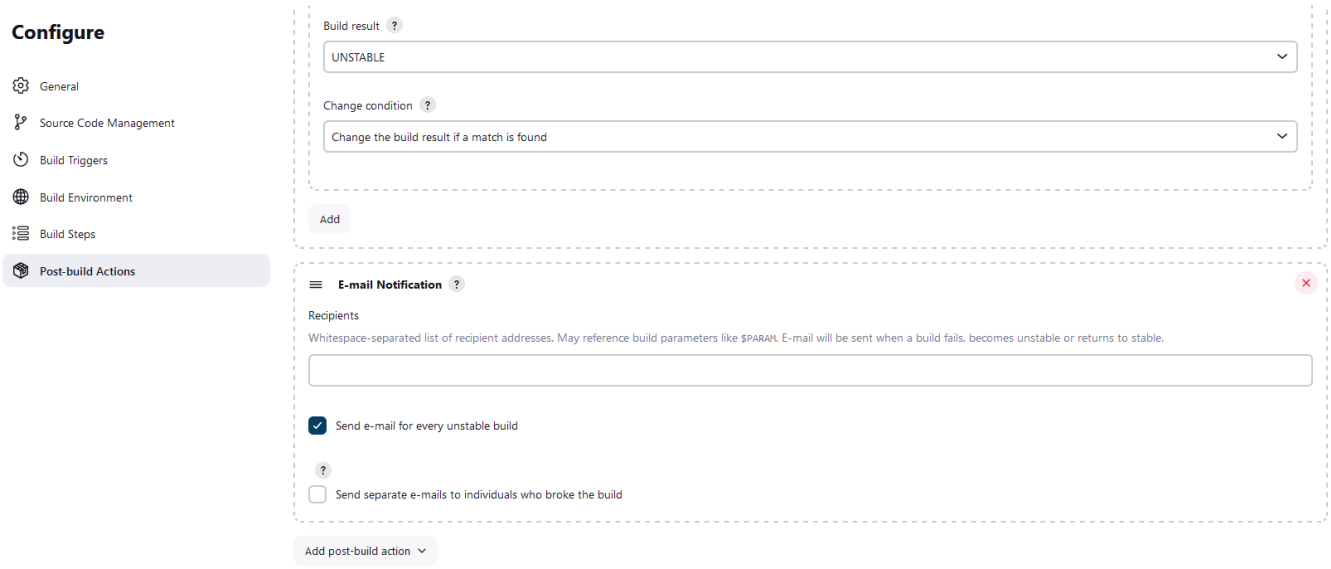
If you change the parameter `exportCommitHistory` to false, the build will become unstable because of this configuration. So you should deactivate the option "Unstable if found" to avoid this warning that does not make sense with this parameter set to false. Don't forget to set it back if you set the value to true again.

Add e-mail notification on failed backup

Thanks to the Jenkins Text Finder post-build action, if the logs of a build contain the text **Warning**, the build is marked as unstable (with a yellow icon). You can go further and be notified by email in that case. In the **Project - Import** configuration page, scroll down or select the tab **Post-build Actions**. There, click on the **Add post-build action** button and choose **E-mail notification**.



On this new action, you just need to add the e-mails to be notified in case of unstable build.



How to set the password in secure storage

The importer does not use the same credentials as the user. It is stored in a different entry in the Eclipse 'Secure Storage'. Storing and clearing the credentials requires a dedicated application that can be executed as an Eclipse Application or using a Jenkins job.

Examples

example1: import project

```
importer.bat -nosplash -data importer-workspace
```

```
-closeServerOnFailure true
-backupDbOnFailure true
-outputFolder C:/TeamForCapella/capella/result
-connectionType ssl
-checkSize 10
```

example2: import user profile model

```
importer.bat -nosplash -data importer-workspace
-closeServerOnFailure false
-backupDbOnFailure false
-outputFolder C:/TeamForCapella/capella/result
-connectionType ssl
-checkSize -1
-importType SECURITY_ONLY
```

4.4. Exporter Configuration

The exporter is an application used to export all projects from a given local folder into a remote repository. It can also be used to export the user profiles model.

Exporter strategy

The Exporter application supports one strategy :

- *Connected export*: the Exporter application establishes a connection to the targeted repository and export projects (chosen by the user).
  - Credentials might be required if the server has been configured to use identification, authentication or user profiles, see Server Configuration job documentation.

See also Projects — Export job documentation.

Exporter parameters

exporter.bat file uses **-vmargs** as a standard eclipse parameter. Eclipse parameters that are used by *exporter.bat* override the value defined in *capella.ini* file. So if you want to change a system property existing in *capella.ini* (-vmargs -Xmx3000m for example) do not forget to do the same change in *exporter.bat*.

The exporter needs credentials to connect to the CDO server if the server has been started with authentication or user profile. Credentials can be provided using either `-repositoryCredentials` or `-repositoryLogin` and `-repositoryPassword` parameters. Here is a list of arguments that can be set to the Exporter (in exporter.bat or in a launch config):

Arguments	Description
-repositoryCredentials	Login and password can be provided using a credentials file. It is the recommended way for confidentiality reason. If the credentials file does not contain any password, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage This parameter must not be used with `-repositoryLogin` or `-repositoryPassword` parameters else the exporter will fail. To use this property file • Add the following program argument: `-repositoryCredentials <path_to_credentials_file>` • Fill the specified file using the following format (only one line allowed): `aLogin:aPassword`
-repositoryLogin	The exporter needs a login to connect to the CDO server if the server has been started with authentication or user profile. `-repositoryLogin` must not be used with `-repositoryCredentials` else the application will fail.

Arguments	Description
-repositoryPassword	This parameter is used to provide a password to the exporter accordingly to the login. If `-repositoryPassword` is not used, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage `-repositoryPassword` must not be used with `-repositoryCredentials` else the application will fail.  Some special characters like double-quote might not be properly handled when passed in argument of the exporter. The recommended way to provide credentials is through the repositoryCredentials file or the secure storage.
-hostname	Define the team server hostname (default: localhost).
-port	Define the team server port (default: 2036).
-connectionType	The connection kind can be set to **tcp** or **ssl** (keep it in low-case) (default: tcp)
-repoName	Define the team server repository name (default: repoCapella).
-sourceToExport	Define the path of folder containing projects to export. This folder can be : • a folder that contains one or more projects to export, • a zip containing one or more Sirius project that is aird file, • a folder that contains one or more zip files.
-logFolder	Define the folder where to save logs (default: -outputFolder).
-overrideExistingProject	If the remote repository already contains a project to export with the same name, this argument allows removing this existing project (default: false).

Arguments	Description
-mergeDifferenceOnExistingProjects	If -overrideExistingProject is set to true (default: false), this argument allows selecting one of the two following override strategies: • Replace: Delete remote resources content and replace by local content (commit history is lost) (default) • Merge: Use Diff/Merge to compare local and existing resources and commit only the differences.
-overrideExistingImage	If the remote repository already contains an image with the same name, this argument allows ignoring and overriding it.
-closeServerOnFailure	Ask to close the server on project export failure (default: false). If the server hosts several repositories, it is better to use the parameter -stopRepositoryOnFailure.
-stopRepositoryOnFailure	Ask to close the repository on project export failure (default: false).  It is currently not possible to restart a single repository, if defined in *cdo-server.xml*. To restart the stopped repository, stop and restart the server.
-addTimestampToResultFile	Add a time stamp to result files name (.zip, logs, commit history) (default: true).
-httpLogin	Exporter application will trigger an Http request. This argument allows giving a login to identify with on the Jetty server.
-httpPassword	Exporter application will trigger an Http request. This argument allows giving a password to authenticate with on the Jetty server.
-httpPort	Exporter application will trigger an Http request. This argument allows giving a port to communicate with on the Jetty server.
-httpsConnection	Exporter application will trigger an Http request. This boolean argument specifies if the connection should be Https or Http.
-help	Print help message.

If the server has been started with user profile, the Exporter needs to have write access to the whole repository (including the user profiles model). See Resource permission pattern examples section.

If this recommendation is not followed, the Exporter might not be able to override existing projects on remote, for example. This may lead to a failed export.

The exporter uses the default configuration of Capella and does not need its own configuration area. For this to work properly, the exporter needs to have read/write permission to the configuration area of Capella, otherwise it can end up with some errors about access being denied. A common situation where the exporter can be found in this situation is when the Scheduler is launched as a Windows service. In this case, the user account executing the service is not necessarily configured to have the read/write permission to Capella's configuration area. If somehow you cannot give the read/write permission to the exporter, a workaround is to provide it a dedicated configuration area by adding the following arguments at the end of exporter.bat file: **-Dosgi.configuration.area="path/to/exporter/configuration/area"** and if necessary, update the existing argument **-data exporter-workspace** to point to a location with read/write permission.

How to set the password in secure storage

The exporter does not use the same credentials as the user. It is stored in a different entry in the Eclipse 'Secure Storage'. Storing and clearing the credentials requires a dedicated application that can be executed as an Eclipse Application or using a Jenkins job.

Examples

example1: export project

```
exporter.bat -nosplash -data exporter-workspace
-closeServerOnFailure true
-connectionType ssl
-sourceToExport C:\Users\me\Documents\runtime-T4C
```

4.5. Connection Application Configuration

The connection application is used to create a local project connected to a shared project on a remote repository. It uses the same processes as the connection wizard.

Connection application strategy

The connection application supports one strategy :

- *Connection*: the connection application establishes a connection to the targeted repository,

checks that the targeted shared project exists and then create a local project that references the remote project.

- Credentials might be required if the server has been configured to use identification, authentication or user profiles, see Server Configuration job documentation.

Connection application parameters

connect.bat file uses **-vmargs** as a standard eclipse parameter. Eclipse parameters that are used by *connect.bat* override the value defined in *capella.ini* file. So if you want to change a system property existing in *capella.ini* (-vmargs -Xmx3000m for example) do not forget to do the same change in *connect.bat*.

The connection application needs credentials to connect to the CDO server if the server has been started with authentication or user profile. Credentials can be provided using either `-repositoryCredentials` or `-repositoryLogin` and `-repositoryPassword` parameters. Here is a list of arguments that can be set to the connection application (in connect.bat or in a launch config):

Arguments	Description
-repositoryCredentials	Login and password can be provided using a credentials file. It is the recommended way for confidentiality reason. If the credentials file does not contain any password, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage This parameter must not be used with `-repositoryLogin` or `-repositoryPassword` parameters else the connection application will fail. To use this property file • Add the following program argument: `-repositoryCredentials <path_to_credentials_file>` • Fill the specified file using the following format (only one line allowed): `aLogin:aPassword`

Arguments	Description
-repositoryLogin	The connection application needs a login to connect to the CDO server if the server has been started with authentication or user profile. `-repositoryLogin` must not be used with `-repositoryCredentials` else the application will fail.
-repositoryPassword	This parameter is used to provide a password to the connection application accordingly to the login. If `-repositoryPassword` is not used, the password will be searched in the eclipse secure storage. See how to set the password in the secure storage `-repositoryPassword` must not be used with `-repositoryCredentials` else the application will fail.  Some special characters like double-quote might not be properly handled when passed in argument of the connection application. The recommended way to provide credentials is through the repositoryCredentials file or the secure storage.
-hostname	Define the team server hostname (default: localhost).
-port	Define the team server port (default: 2036).
-connectionType	The connection kind can be set to **tcp** or **ssl** (keep it in low-case) (default: tcp)
-repoName	Define the team server repository name (default: repoCapella).
-remoteAirdPath	Define the path specifying the remote aird to connect with (using "/" and not "\").
-localProjectName	Define the name of the local project to create.
-deleteLocalProjectOnError	Define the Boolean value to declare if the application should delete the local project if an error is detected (False by default).
-logFolder	Define the folder where to save logs (default: -outputFolder).

Arguments	Description
-addTimestampToResultFile	Add a time stamp to result files name (.zip, logs, commit history) (default: true).
-help	Print help message.

How to set the password in secure storage

The connection application does not use the same credentials as the user. It is stored in a different entry in the Eclipse 'Secure Storage'. Storing and clearing the credentials requires a dedicated application that can be executed as an Eclipse Application or using a Jenkins job.

Examples

example1: creation of local project "capella_test_local" in workspace "connection-workspace" connected to shared project "capella_test" on default location (localhost:2036/repoCapella)

```
connect.bat -nosplash -data connection-workspace
-remoteAirdPath /capella_test/capella_test.aird
-localProjectName capella_test_local
```

example2: creation of local project "capella_test_local" in workspace "connection-workspace" connected by WSS protocol to shared project "capella_test" on server capella.mydns.com through port 2037 and repository named "CapellaSharedProjects"

```
connect.bat -nosplash -data connection-workspace
-connectionType wss
-hostname capella.mydns.com
-port 2037
-repoName CapellaSharedProjects
-remoteAirdPath /capella_test/capella_test.aird
-localProjectName capella_test_local
```

4.6. Client preferences initialization

Introduction

As any eclipse application, Team For Capella uses preferences to manage the behavior of the application.

There are many preference scopes, including the default and the instance scope. Instance scope, if set, has the priority to the default scope. The default scope is the value by default provided by the application. The instance scope corresponds to the preferences a user can change with the **Preferences dialog box** accessible with the menu Windows/Preferences. These preferences are stored in the user's workspace. For more details, refer to the eclipse Preferences documentation

For more information about the preferences used for Team For Capella, refer to the client preferences documentation.

The Administrator, in charge of customizing the product functionalities, may want to

- either **set the default value for the preferences** for the application. (recommended)
- or set the preference for the workspace and export it as an epf file.

Setting the default preference values (recommended)

To initialize the default preferences without having to provide a plug-in, you can use the pluginCustomization Eclipse parameter. Refer to Eclipse Runtime documentation for more information.

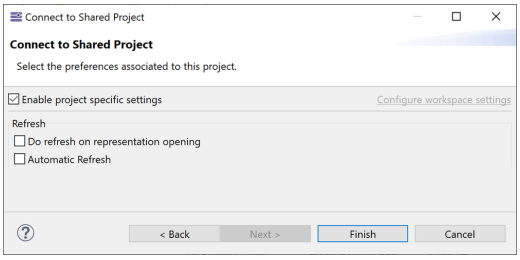
The principle is to declare a property file which contains pairs of key/value. The key is the qualified name of the preference, and the value is the value of the preference.

- The capella.ini file, next to the Capella executable file, should contain the line **-pluginCustomization pluginCustomization.ini**. If not, add it before vmargs arguments.
- Then in **pluginCustomization.ini**, add <plugin_name>/<preference_name>=<value>

Preference keys

Preferences have a default value associated with the Team for Capella application. This chapter explains how to change their default value. Nevertheless, the user has the ability to use a different value, than the default one, using the Preferences dialog box. This will set a value for the scope corresponding to the user workspace. The workspace scope has a higher priority than the default scope.

Sirius Preferences	**Preference keys**	**Default value if not set**
Sirius "Automatic Refresh" and "Do refresh on representation opening"	org.eclipse.sirius.ui/PREF_REFRESH_ON_REPRESENTATION_OPENING=<boolean value> org.eclipse.sirius/PREF_AUTO_REFRESH=<boolean value>	true

Team collaboration Preferences	Preference keys	Default value if not set
Check by default the check button in the "Capella Connected Project" wizard to have the Sirius Refresh preferences specific to the connected project that is being created. 	fr.obeo.dsl.viewpoint.collab/PREF_ENABLE_PROJECT_SPECIFIC_SETTINGS_DEFAULT_VALUE =<boolean value>	true
Connection Url 1- Alias 2- Server IP address 3- Server port 4- Connection type 5- Repository name	1- fr.obeo.dsl.viewpoint.collab/PREF_DEFAULT_REPOSITORY_ALIAS=<string value> 2- fr.obeo.dsl.viewpoint.collab/PREF_DEFAULT_REPOSITORY_LOCATION=<string value> 3- fr.obeo.dsl.viewpoint.collab/PREF_DEFAULT_REPOSITORY_PORT=<integer value> 4- fr.obeo.dsl.viewpoint.collab/PREF_DEFAULT_CONNECTION_TYPE= enumeration [TCP, SSL] 5- fr.obeo.dsl.viewpoint.collab/PREF_DEFAULT_REPOSITORY_NAME=<string value>	1- "Default" 2- localhost 3- 2036 4- TCP 5- repoCapella

Team collaboration Preferences	Preference keys	Default value if not set
Commit history view 1- Require description for commit actions 2- Pre-fill commit description 3- Commit description provider 4- Automatically use the pre-filled description when none is provided	1- fr.obeo.dsl.viewpoint.collab/PREF_ENABLE_DESCRIPTION_ON_COMMIT=<boolean value> 2- fr.obeo.dsl.viewpoint.collab/PREF_COMPUTE_COMMIT_DESCRITION=<boolean value> 3- fr.obeo.dsl.viewpoint.collab/PREF_PREFERRED_DESC_PARTICIPANT=complex value 4- fr.obeo.dsl.viewpoint.collab/PREF_AUTO_USE_PRE_FILLED_COMMIT_DESC=<boolean value>	1- false 2- false 3- Default 4- false
Release all explicit locks after committing	fr.obeo.dsl.viewpoint.collab/PREF_RELEASE_EXPLICIT_LOCK_ON_COMMIT=<boolean value>	false
Display Write Permission Decorator	fr.obeo.dsl.viewpoint.collab/PREF_DISPLAY_WRITE_PERMISSION_DECORATOR=<boolean value>	true
Ability to lock the semantic element at representation creation or move	fr.obeo.dsl.viewpoint.collab/PREF_LOCK_SEMANTIC_TARGET_AT_REPRESENTATION_LOCATION_CHANGE=<boolean value>	true

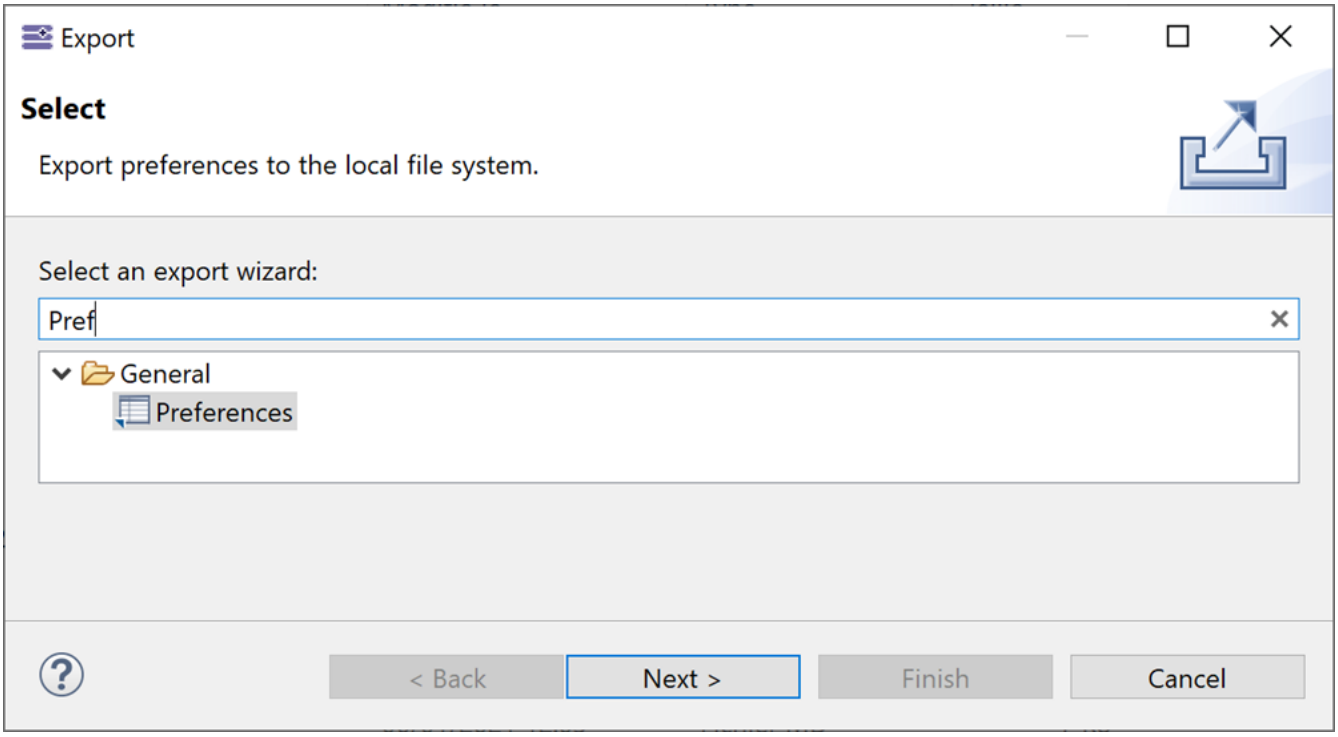
How to discover the preference value

Sometimes, the value of the preference is complex. It is the case for some preferences visible in the **Preferences** dialog box. To know the value of a particular preference:

- change the preference with the Preferences dialog box
- exit eclipse
- check the value in the files <workspace_name>/.metadata/.plugins/org.eclipse.core.runtime/<plugin_name>.prefs

Setting the preferences value for the workspace

Once you have configured the preferences using the Preference dialog box, you have to export the preferences to a text file:



Then each user will have to import the preference file to set the preferences values for their workspace.

- The import process has to be done for each workspace.
- Using the Preference dialog box allows you to configure the preferences without knowing the technical name of the preferences, but some preferences are not available in the Preference dialog box. So you have to add it manually in the exported preferences file. Refer to the Preference keys to know what to add in the preference file.

Chapter 5. System Administrator Guide

Contents

5.1. Overview

System administrators handle the installation, configuration and authentication on the CDO server that is used for sharing Capella projects. For these activities, Team for Capella provides the following functionalities in Eclipse or as jobs which can be installed in a Jenkins used as a scheduler:

- Jenkins Installation
  - For an easy management of the CDO server (Start, stop, list users…) and the shared projects (Backup, diagnostics…), Team for Capella provides many applications ready to deploy in Jenkins.
- Storage on a shared server
  - Team for Capella runs on a server shared across all your authorized team members. It can be administered to properly start and stop the system, and see who is currently connected. Models can be stored on one or several database(s) deployed on one or several machine(s).
- Server Administration
  - Once a server has been configured, there are administration features to manage it while running, such as durable lock management and user management with dedicated Eclipse view. Furthermore, there are also jobs for diagnostic and repair available on the Jenkins Interface.
- Secured access
  - Definition of authorized users and roles stored in a model on the server
  - Data stored in the repositories can be protected by using LDAP to authenticate users, and by using SLL to encrypt the exchanges between the clients and the database(s). It is also possible to define access rights depending on user profiles.
- **Flexible licensing mode**
  - Our floating licensing mode allows you to deploy Team for Capella in a flexible way, depending on your context and your infrastructure: licenses are floating, allowing them to be shared among several users over time, when required due to low network's bandwidth, remote desktop mechanism is supported, avoiding you to deploy Team for Capella client on user's machines, large organizations working with Capella on several projects can deploy Team for Capella server on several machines simultaneously: the licensing mode only

controls the number of current connected users, not the number of running servers.

Team for Capella bundles and installation guide are available at https://www.obeosoft.com/en/team-for-capella-download.

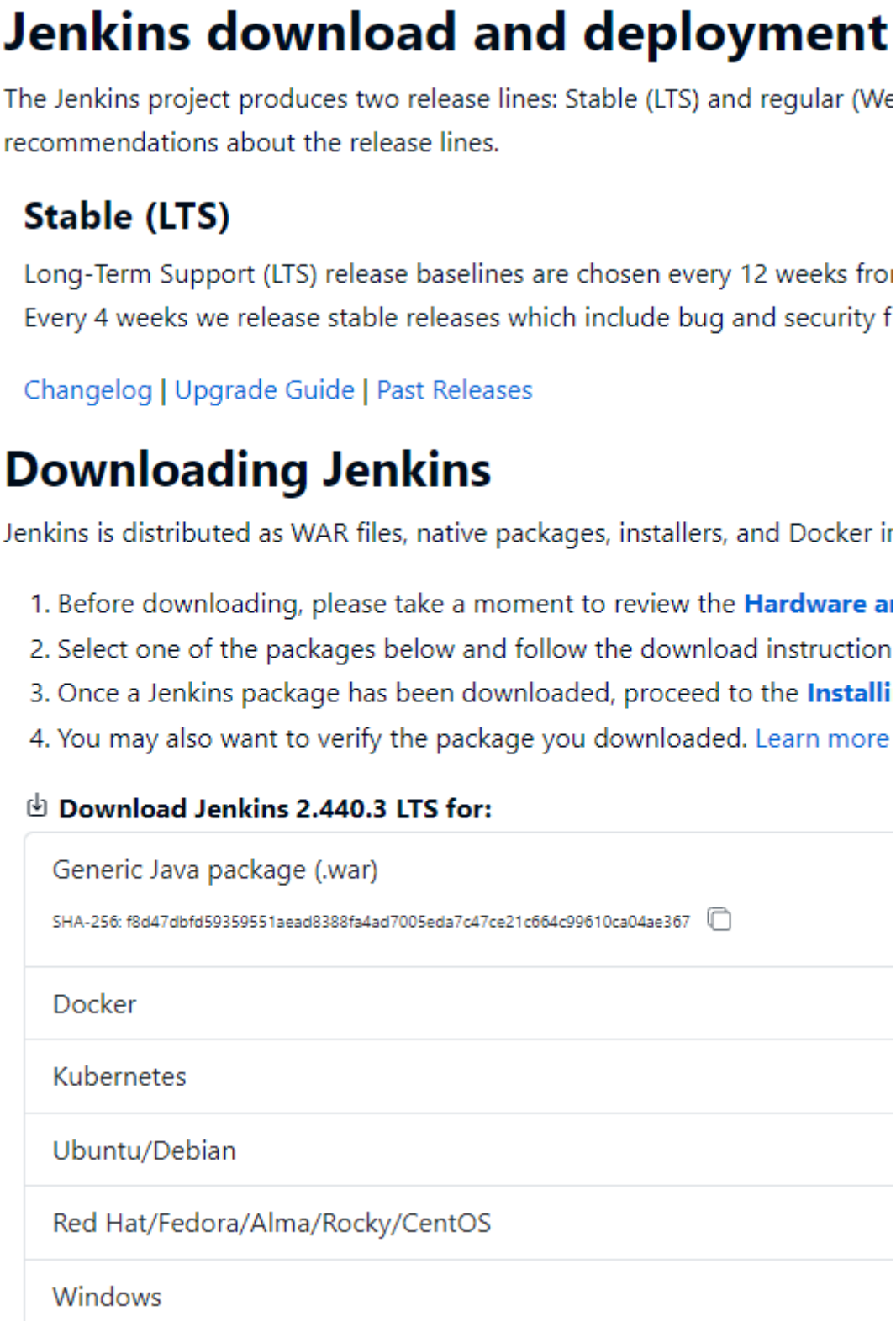
5.2. Jenkins Installation

The documentation of Team for Capella presents many applications (Backups, diagnostics...) that can be scheduled with Jenkins in order to have a centralized platform to manage your shared projects.

Download and install Jenkins

It is recommended to install a 2.440.3 LTS release. Team for Capella 7.0.0 has been tested with Jenkins 2.440.3 LTS release.

If you choose to deploy a more recent version, we strongly recommend to use a release from the LTS (Long Term Support) stable releases stream available at Jenkins.io.



The default Jenkins port is 8080. But **it is recommended to set the port to 8036** (In the previous Team for Capella installation, the embedded Jenkins was deployed on port 8036). Otherwise, there will be a conflict with the REST admin server which default port is 8080.

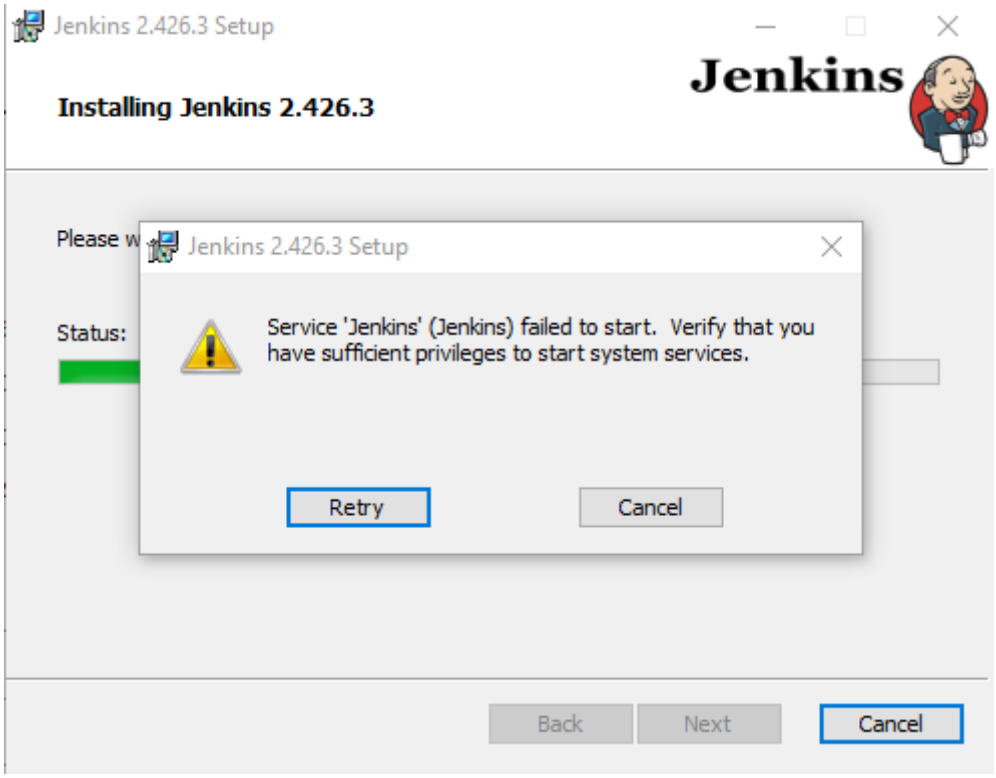
The port can be chosen in the Jenkins installation wizard. This following documentation will often reference the port 8036.

Windows

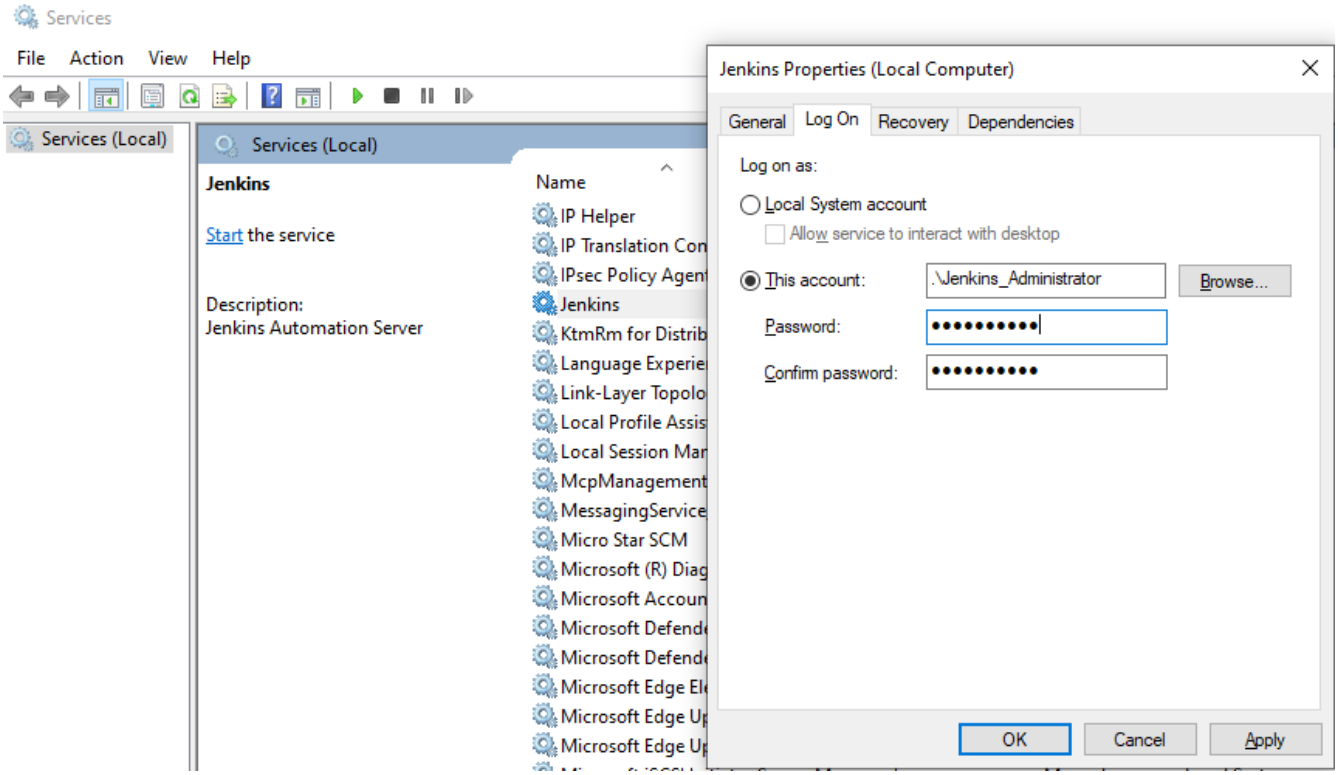
The Jenkins 2.440.3 LTS Windows installer can be downloaded from this link.

If you choose to deploy a more recent version, we strongly recommend to use a release from the LTS (Long Term Support) stable releases stream available at Jenkins.io.

Once downloaded, proceed to the installation. It is recommended to install the Jenkins service (automatic loading on restart). Note that if you previously had an older version of Jenkins on this machine, with the same user account, even if you uninstall it properly you may have the installation process stopping with this message:

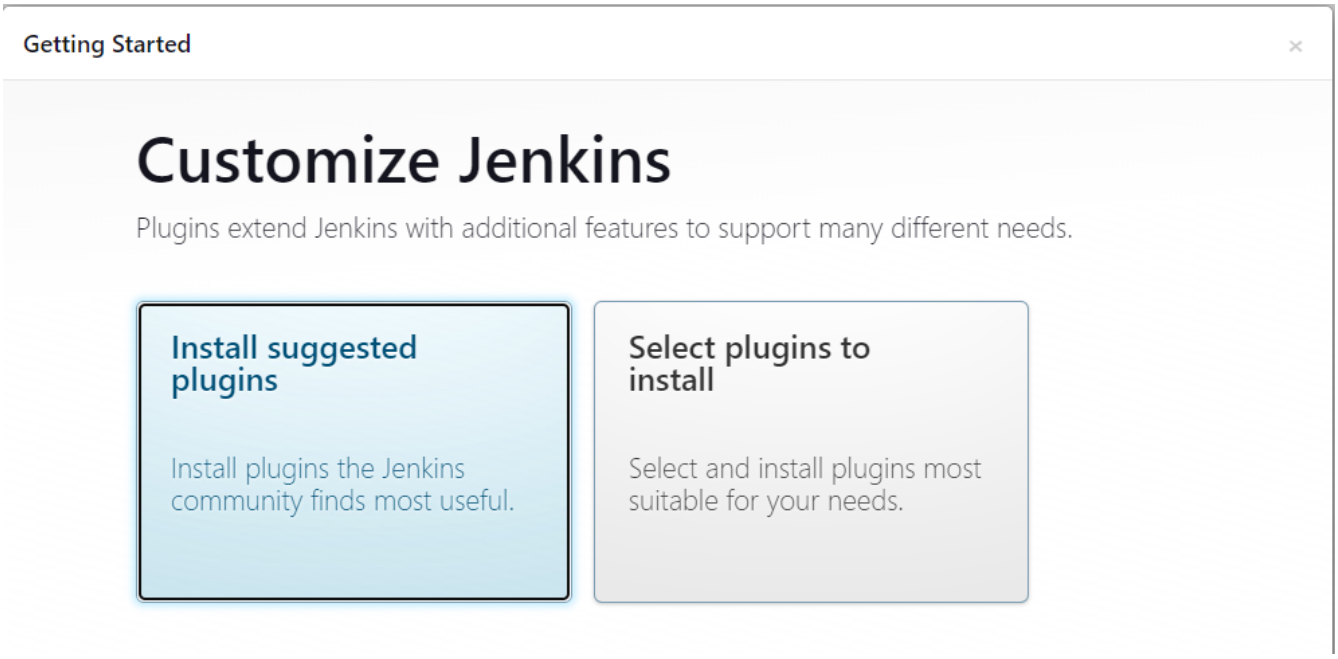


In that case, you need to open your service menu, find the existing Jenkins service. Go to its properties, tab "Log On" and set the password again.

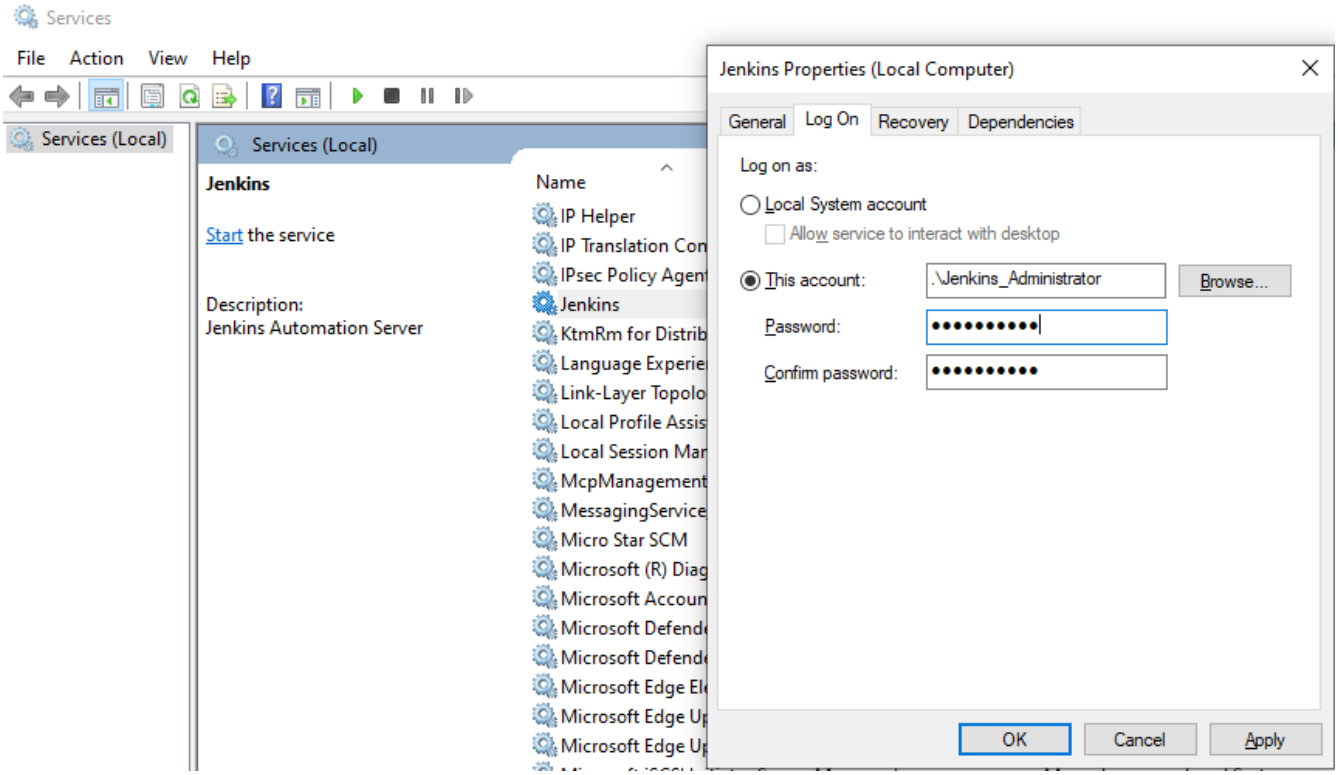


Once the installation is done, you can enter the address **localhost:8036** to initialize the Jenkins application.

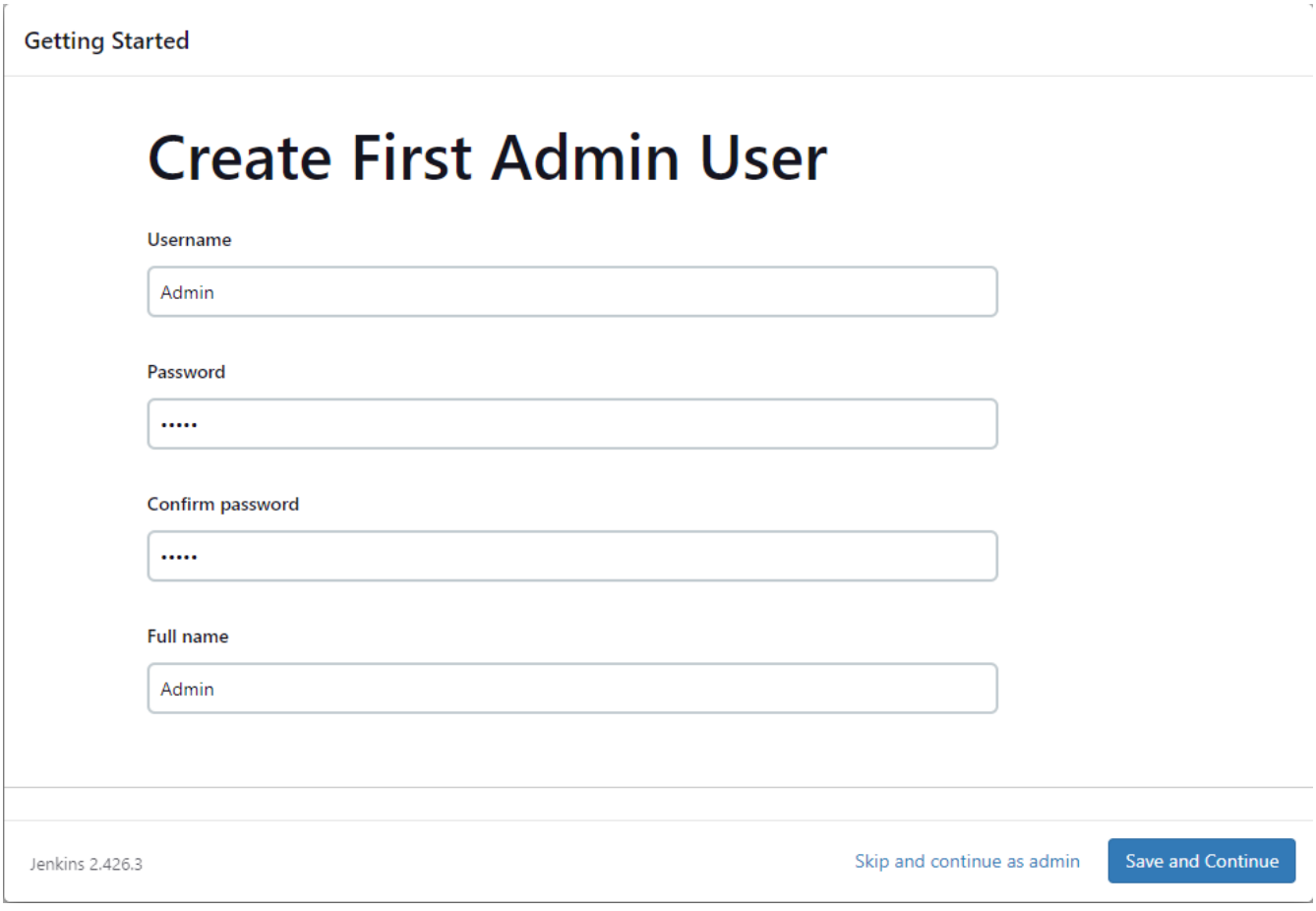
After an authentication page asking to find a generated password, there is a customization page suggesting some plugin installation.



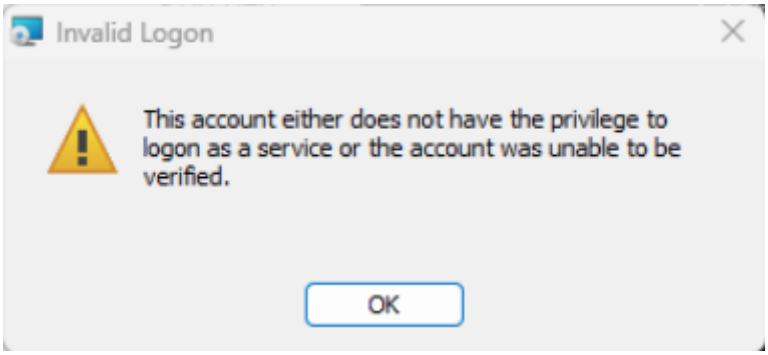
This step can be ignored. In the next step of this guide, you will execute a script that install all the Team for Capella jobs in Jenkins and the necessary plugins. You can press on the top-right cross, but that will end the initialization wizard, skipping a step where you can configure the admin password. We recommend to click on the "Select plugins to install" button. On the next page, select **None** and then the **Install** button.



Finally, create your admin login/password. This will replace the generated password that was used at the begining of this wizard. Click on **Save and Continue** to complete this wizard.



Note: If the Jenkins installation on Windows 10/11 Home is blocked despite using an administrator account (See popup below), it can be a configuration issue preventing access to the Local Group Policy Editor (gpedit.msc).



This editor can be activated with two commands using a command prompt as an administrator: * FOR %F IN ("%SystemRoot%\servicing\Packages\Microsoft-Windows-GroupPolicy-ClientTools-Package~**.mum") DO (DISM /Online /NoRestart /Add-Package:"%F") * FOR %F IN ("%SystemRoot%\servicing\Packages\Microsoft-Windows-GroupPolicy-ClientExtensions-Package~**.mum") DO (DISM /Online /NoRestart /Add-Package:"%F")

After this, restart the Jenkins installation.

Linux

The Jenkins 2.440.3 LTS packages for Linux can be downloaded from the LTS Releases package repository corresponding to the targeted distribution. See this link.

The scheduler has been tested on RedHat and Debian based distributions. The Jenkins installation instructions are available at Installing Jenkins: Linux

The command line tools launch the Capella runtime and it requires a display even when started in an headless way. An Xvnc server needs to be installed on the Linux server:

- On Debian based distributions, you can install either tigerVNC or TightVNC:

```
sudo apt install tightvncserver
```

```
sudo apt install tigervnc-standalone-server
```

- On RedHat based distributions:

```
dnf install tigervnc-server
```

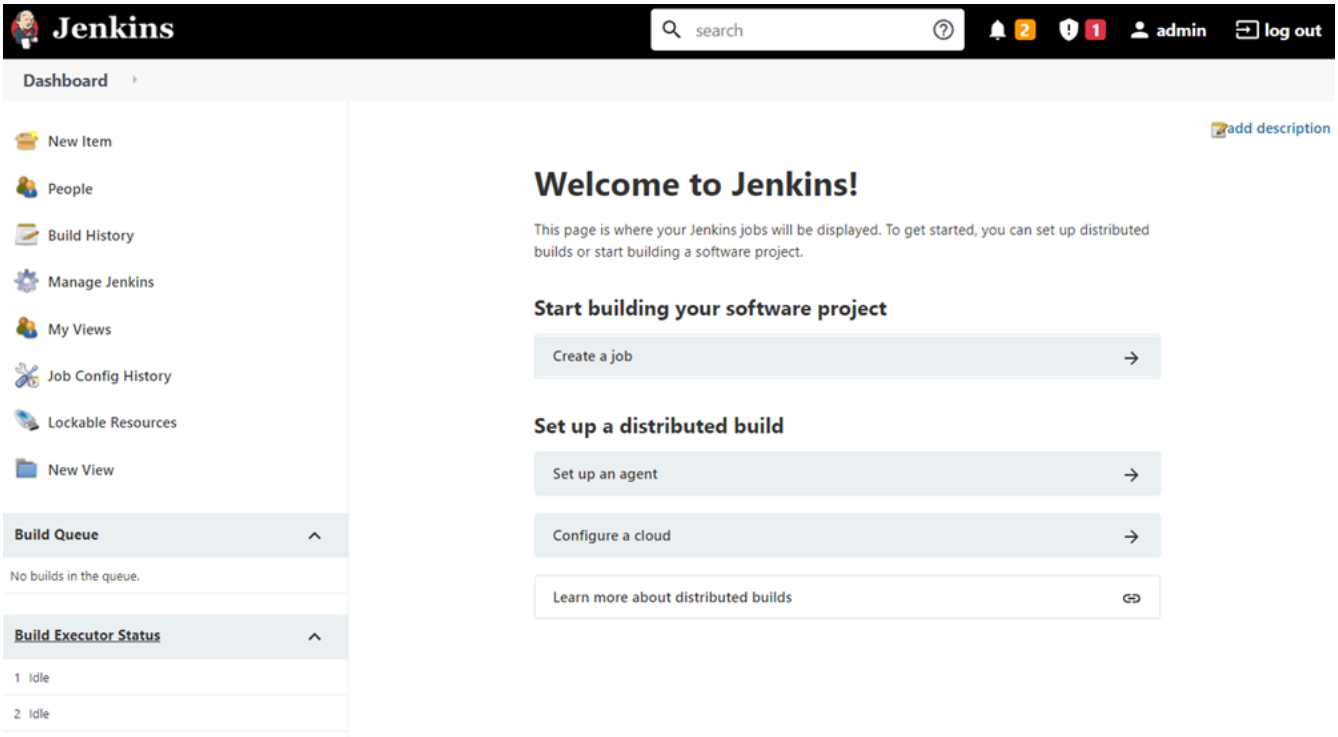
In addition, make sure that the Xvnc jenkins plugin is installed on the Jenkins (it is installed by install-TeamForCapellaAppsOnJenkins.sh).

Make sure that the *jenkins* user has read, write and execution permission on the TeamForCapella root folder.

```
chmod -R 770 jenkins:jenkins path_To_T4C_Installation_Folder
```

End of the installation

At the end of the installation, your web browser should be displaying Jenkins.



Install Jenkins plugins and jobs required for Team for Capella

Automatic installation

Once Jenkins is installed, you can run our installation script that will install all the jobs allowing the Jenkins scheduler to manage the different Team for Capella applications. This script also downloads all the Jenkins plugins required for the different jobs.

In your Team for Capella installation folder, go to the *tools/resources/scheduler* folder. In this folder, you will find the installation script: **install-TeamForCapellaAppsOnJenkins.bat** (or install-TeamForCapellaAppsOnJenkins.sh for Linux). It contains all the required commands to download and install the plugins, and install and configure the jobs.

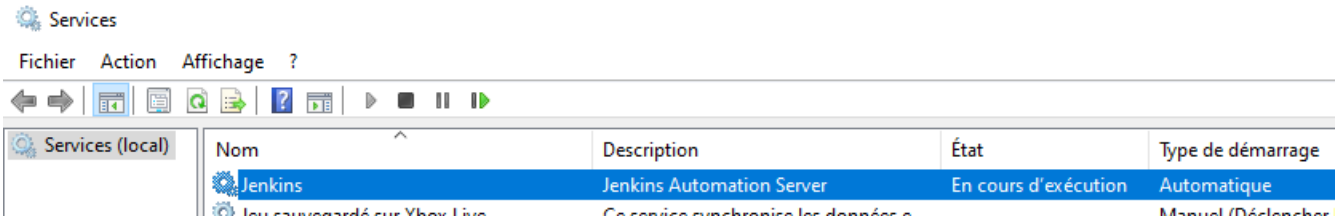
It comes with **install-TeamForCapellaAppsOnJenkins.properties**, a properties files containing parameters to configure the script commands:

- **JENKINS_URL**: The web address of your Jenkins
- **JENKINS_USER_ID**: The login of a user able to connect to Jenkins
- **JENKINS_API_TOKEN**: The password or API token of this user
- **INSTALL_T4C_SCHEDULER_JOBS_AND_VIEWS**: A boolean allowing to skip or run the jobs and views installation steps.
- **INSTALL_T4C_SCHEDULER_REQUIRED_JENKINS_PLUGINS**: A boolean allowing to skip or run the Jenkins plugins installation steps.
- **PLUGINS_LIST_FILE**: the name of the text file containing the list of plugins. Four predefined files:

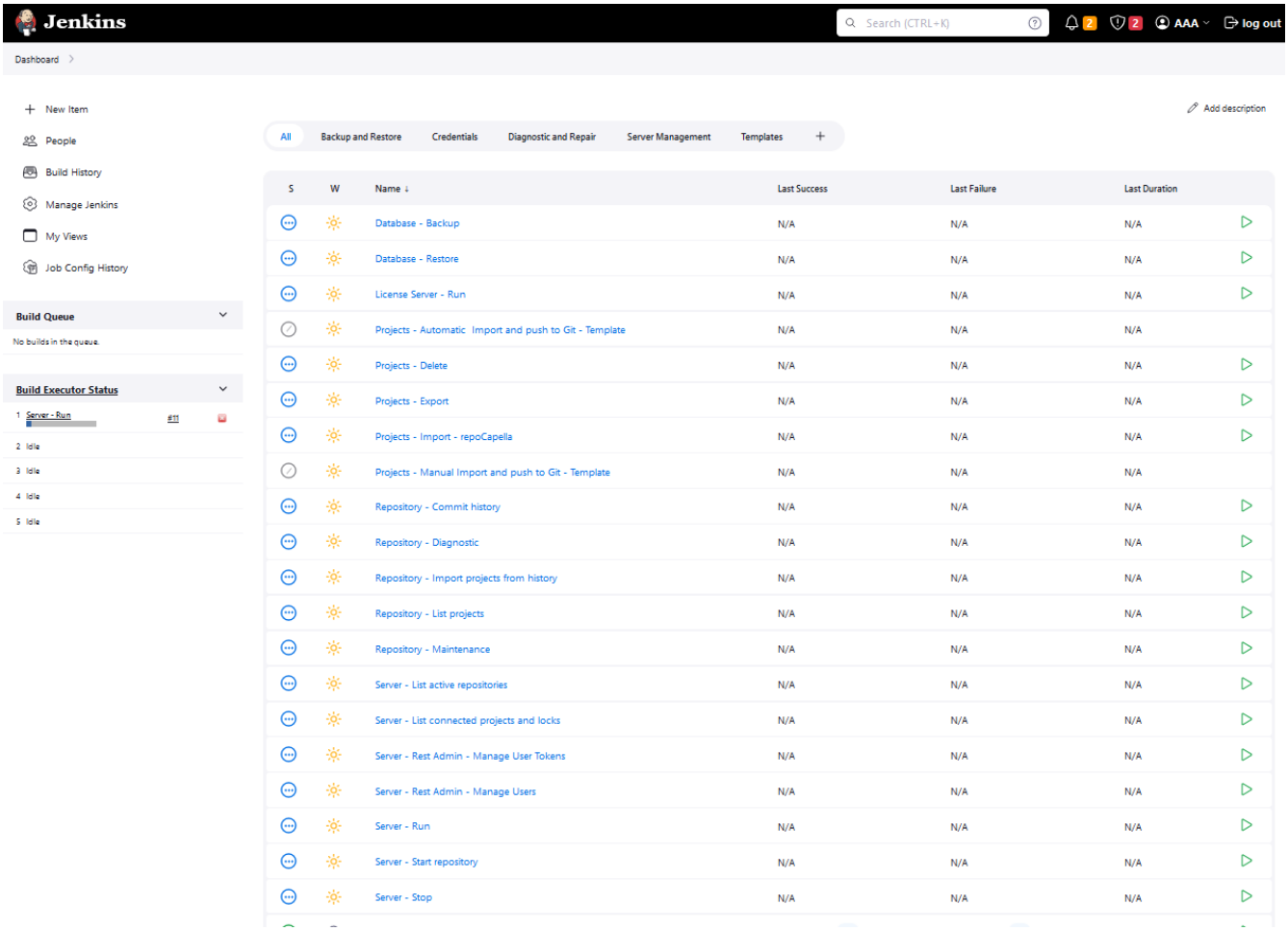
- the first one with a list of plugins versioned urls (default, exact tested version for Team for Capella 7.0.0 and Jenkins LTS 2.440.3),
- the second one with a list of plugin short names. Plugins will be installed from the update center,
- the third and fourth lists correspond to the list of plugins which were listed in T4C 6.1 and previous versions.

- **JAVA_CALL**: The path to the java executable to use for the installation. By default it points to the JRE embedded in Capella.

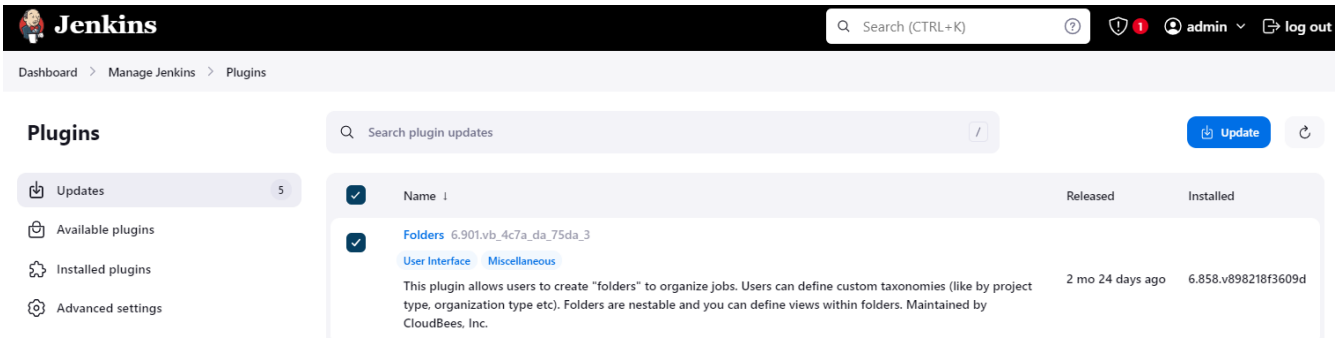
As documented in https://www.jenkins.io/doc/book/managing/cli/, you can get your API token from */me/configure* page of your Jenkins. The script will automatically download the Jenkins CLI client and use it to install the plugins. Then it will create all the Team for Capella jobs and sort them into different views. Finally, once the script finished, you only need to restart Jenkins. The simplest way is to use the */restart* page of your Jenkins. On Windows, if you have installed Jenkins, to restart it, you could also use your system **Services** window.



The dashboard will present all the Team for Capella applications.



Note that the plugins versions were chosen at the time of the release of the Team for Capella version you are working on. Once the script executed, it is recommended to keep Jenkins up to date and also to check for new updates of the installed plugins. Go to **Manage Jenkins** > **Plugins**. On the **Update** tab, select all plugins and then click on the **Update** button.



These jobs executes Team for Capella applications, therefore Jenkins requires a global environment variable referencing the location of your team for Capella installation:

- Go to **Manage Jenkins** > **Configure System** and scroll down to the **Global properties** section.
- Check **Environment variables** and add a new one named **TEAMFORCAPELLA_APP_HOME** with the path to your Team for Capella installation folder as the value (it is the top folder that contains the subfolders *capella*, *tools*, ...).

Note that the development team is working on improving the installation script to add this variable, but some Jenkins APIs have been removed for security reasons as it was seen as code injection.

Additional configuration steps are recommended, see Executors, Locale, Default view and Display Job Description in miscellaneous settings section.

Restart Jenkins or its service after this configuration phase.

Manual installation

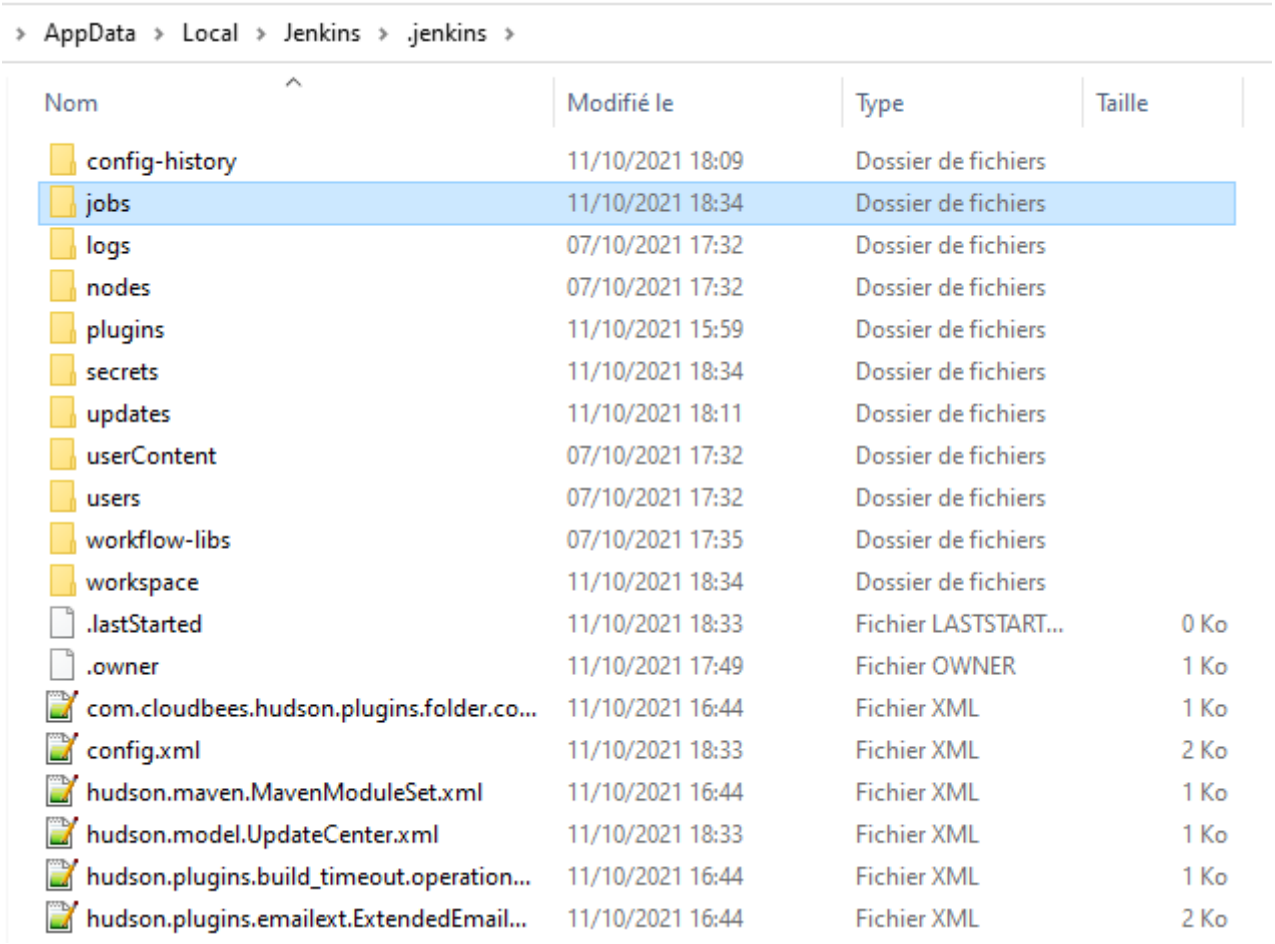
If you do not wish to install the Team for Capella applications with the script, you can still proceed manually. The first step is to install the required plugins. In your Team for Capella installation folder, go to the *tools/resources/scheduler* folder, you will find two files with names starting with *RequiredPlugins*.

They contains the same list of plugins, one lists them by name, the other one list them by URL to their .hpi. You need to install all of them. Go to **Manage Jenkins** > **Manage Plugins** to install them from the plugin manager. Then restart Jenkins.

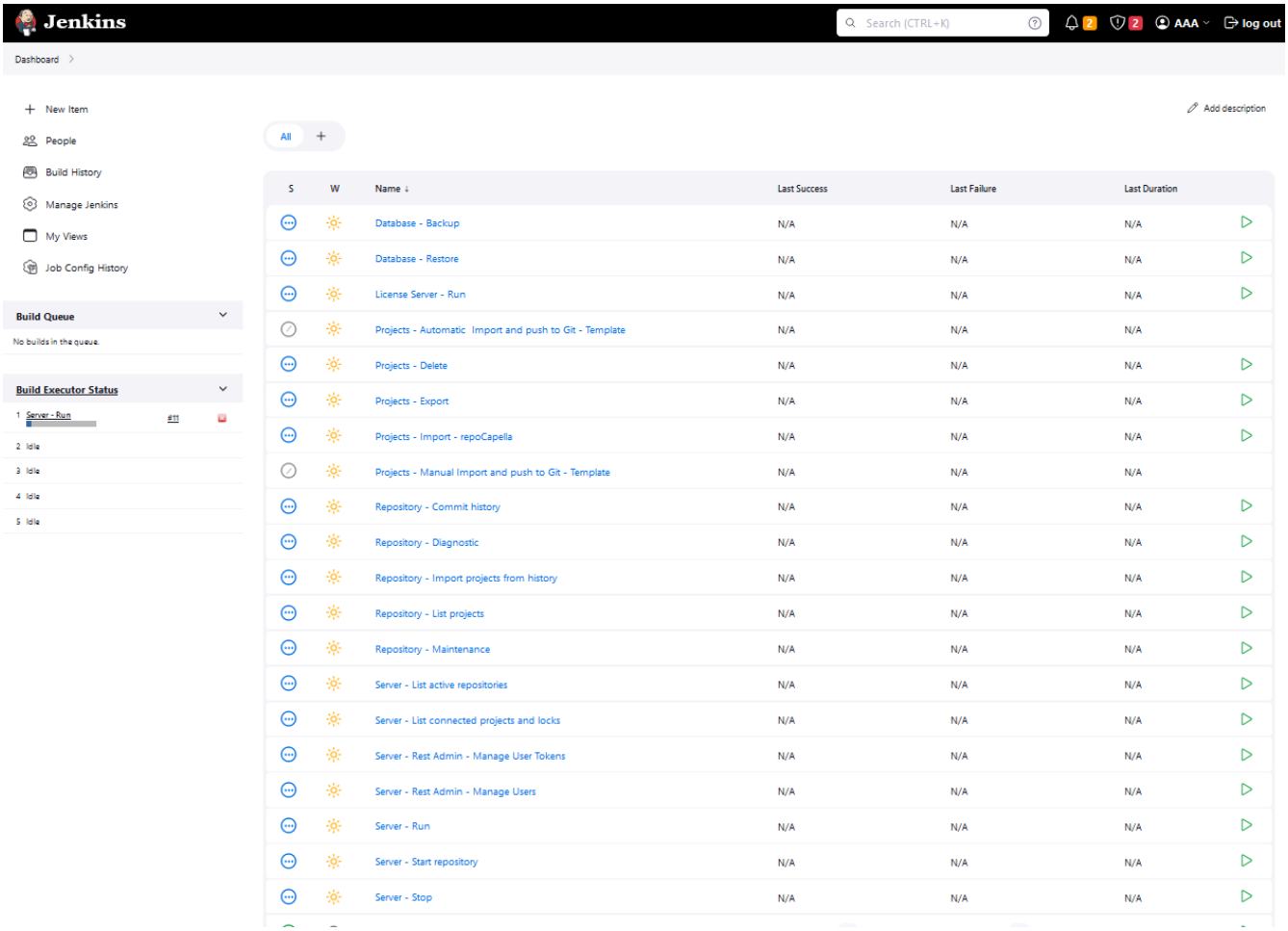
Now that the required plugins have been installed, the Team for Capella jobs can be deployed as well:

- Still in the tools folder of your Team for Capella installation, you can find a folder named **jobs**.

- Copy this folder.
- Then, we will need to paste it in the Jenkins configuration folder.
  - To locate this folder go to Manage Jenkins > Configure System.
  - The first information should be the Jenkins home directory (it should be the user home folder followed by AppData>Locale>Jenkins>.jenkins).
- Go to this folder and paste your clipboard there (there should already be an empty jobs folder that will be fused).-



Restart Jenkins and now the dashboard will present all the Team for Capella applications.



These jobs executes Team for Capella applications, therefore Jenkins requires a global environment variable referencing the location of your team for Capella installation:

- Go to **Manage Jenkins** > **Configure System** and scroll down to the **Global properties** section.
- Check **Environment variables** and add a new one named **TEAMFORCAPELLA_APP_HOME** with the path to your Team for Capella installation folder as the value (it is the top folder that contains the subfolders *capella*, *tools*, ...).

Finally, as there are many jobs, it will be easier to manage by grouping these applications by tabs:

- On the Jenkins dashboard, you can simply press the + button next to the default tab named **All**.
- On the next page, name your tab, select **List view**.
- On the final page, select the jobs that you want on this tab.
- On the same page, you can change the order of the columns.
  - We recommend to move the **Build Button** just after the **Name** column.
  - This way it is easier to trigger a job.

As an example, you can order your tabs as follows:

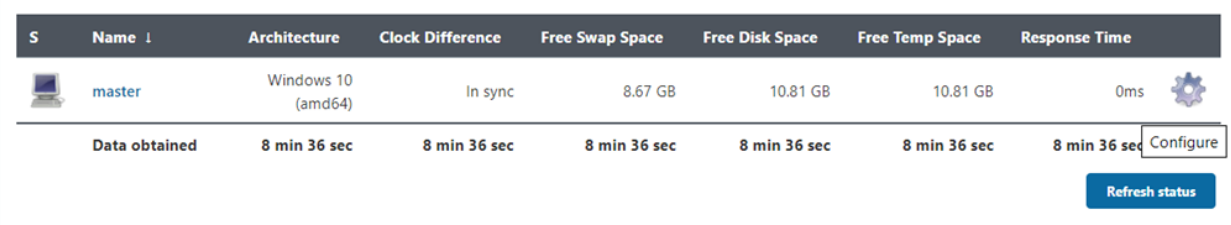
- **Server Management**: Importer – Clear credentials, Importer – Store credentials, License Server – Start, Server – List connected projects and locks on model, Server – Start and Server – Stop.
- **Backup and Restore**: Database – Backup, Database – Restore, Projects – Import and User profile – Import model
- **Diagnostic and Repair**: Repository – Diagnostic and Repository – Maintenance
- **Templates**: Projects – Automatic Import and push to Git – Template and Projects – Manual Import and push to Git – Template

Additional configuration steps are recommended, see Executors, Locale, Default view and Display Job Description in miscellaneous settings section.

Miscellaneous settings

Executors

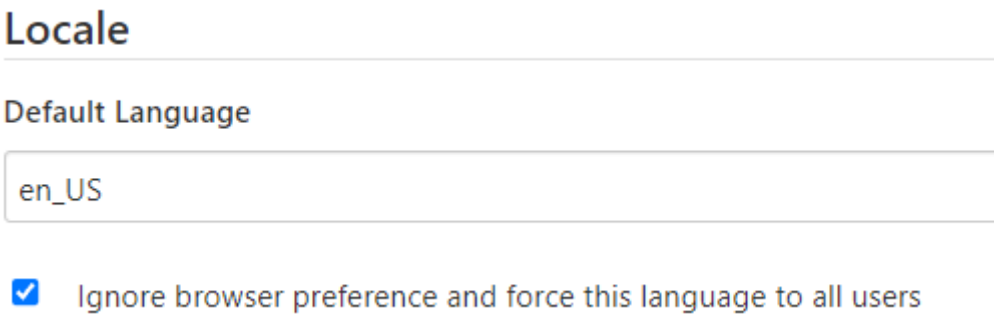
- By default Jenkins provides two build executors. This means that two applications can run at the same time. However, the CDO server and the License server are applications that keeps running. Therefore, they will block any other application. We recommend to change that in order to have five executor. There are two ways to change this:
  - Click on the **Build Executor Status** (Bottom left on the Jenkins Dashboard). Then, on the entry presenting the computer, click on the Configure button. On this executors configuration page, set the number of executors to 5.



  - Go to **Manage Jenkins** > **Configure system**, in the category **Maven Project Configuration** set the variable **# of executors** to 5.

Locale

- By default Jenkins will be presented with the language of the user's system. However, it is possible to force displaying it in a certain language. Go the **Manage Jenkins** > **Configure System**. You then need to locate the **Locale** area, set the chosen language and check the **Ignore browser preference and force this language to all users** checkbox. You can choose the language (for instance *fr* for French) but also the region (for instance *en_US* for American English).



Default view

- As the applications are sorted by views, you can choose a default view (the one shown when you click on the Dashboard) by going to **Manage Jenkins** > **Configure System**, under **Default view** you can choose which one you want to see as default.

Display Job Description

- By default Jenkins will display the job's description as plain text. However the provided jobs have an HTML description. Jenkins configuration can be changed to correclty display such descriptions:
  - Go to **Manage Jenkins** > **Configure Global Security** > **, change the Markup Formatter** from *Plain Text* to *Safe HTML*.

Change the Port Used by Jenkins

Windows

Go to the directory where you installed Jenkins (by default, it's under Program Files/Jenkins), edit **jenkins.xml** , then update the value of *--httpPort* in the <arguments> tag of of the service definition:

```
<executable>java</executable>
<arguments> -some -arguments --httpPort=8036 -some -other - arguments</arguments>
```

Finally, go to Windows service, and restart the **Jenkins** service (or restart the Jenkins server if you launched it manually).

Change the name and id of the Jenkins service

Go to the directory where you installed Jenkins (by default, it's under Program Files/Jenkins), edit **jenkins.xml** , then update the value of the <id> and <name> tags of of the service definition:

```
  <id>TeamForCapellaScheduler</id>
  <name>Team For Capella Scheduler</name>
```

Open a Command Prompt as administrator in this folder and execute the following commands

```
  sc stop jenkins
```

```
sc delete jenkins
jenkins.exe install
jenkins.exe start
```

Finally, go to Windows service, and check that

- the **Jenkins** service is no more present.
- a new service is present with the id and name of your choice.

Linux

The configuration file after a standard installation is located in:

- `/etc/default/jenkins` : for most of the Linux distributions.
- `/etc/sysconfig/jenkins` : for RedHat/CentOS distribution.

By default, the port is 8080:

```
HTTP_PORT=8080
```

The service has to be restarted after the port modification:

```
systemctl restart jenkins
```

Set specific folders for Jenkins

Windows

It is possible to force Jenkins to use some specific folders. Go to the directory where you installed Jenkins (by default, it's under Program Files/Jenkins), edit **jenkins.xml**, then complete the <arguments> tag of the service definition:

- Set the Java temp folder to use in Jenkins: `-Djava.io.tmpdir=%JENKINS_HOME%\temp`
- Use a specific folder to extract the Jenkins war content: `--extractedFilesFolder="%JENKINS_HOME%\temp"`

Finally, go to Windows service, and restart the **Jenkins** service (or restart the Jenkins server if you launched it manually).

Linux

Open the jenkins configuration file (see the previous *Change the Port Used by Jenkins* paragraph for the configuration file location)

- Set the Java temp folder to use in Jenkins: `JAVA_ARGS="-Djava.io.tmpdir=$JENKINS_HOME/temp"`
- Use a specific folder to extract the Jenkins war content by adding the following argument in `JENKINS_ARGS` variable: `--extractedFilesFolder=$JENKINS_HOME/temp"`

Updates

It is recommended to check for updates. On the top-right area, Jenkins will show notifications if there are some updates or issues identified. Furthermore, when you select the **Manage Jenkins** menu, the top area will present updates or corrections that can be applied to Jenkins or its plugins. Depending on the importance it will be presented in different colors (red>yellow>blue). Most of the time, it is notifications about new updates but in any case, it is a good practice to check this page once in a while and follow what is presented.

Uninstall Jenkins

The Jenkins service can be stopped and deleted using the following commands in a Windows Command Prompt:

```
sc stop jenkins
sc delete jenkins
```

The id of the service is *jenkins* by default but you might have changed it as described in Change the name and id of the Jenkins service section.

Jenkins can be completely removed from your system with the use of its Windows Installer.

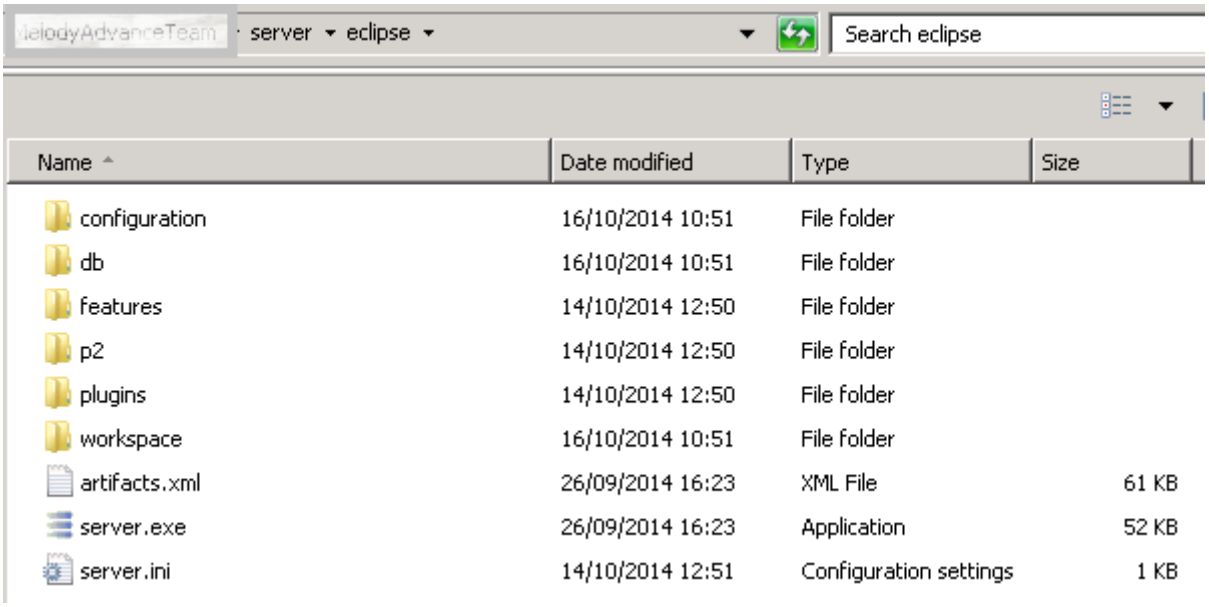
5.3. Server Configuration

In this document you will discover how to manage a Server supporting **Collaborative Modeling** features.

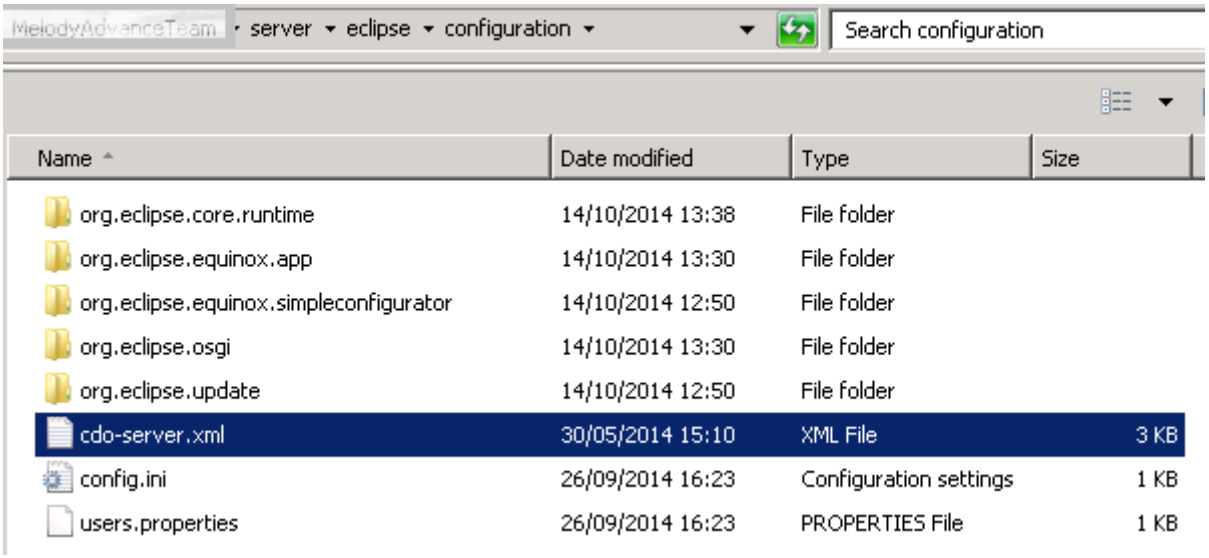
Cdo-server.xml File

The main configuration file used by the Team for Capella Server is the **cdo-server.xml** file.

The Team for Capella Server bundle comes as a standard Eclipse application. In the installed package, locate the Configuration folder and open it.



In this folder, locate the **cdo-server.xml** file and open it.



Here is a commented extract of the "‹cdo-server.xml›" delivered with Team for Capella:


```
<?xml version="1.0" encoding="UTF-8"?>
<cdoServer>
    <!--acceptor type="http"/ -->
    <acceptor type="tcp" listenAddr="0.0.0.0" port="2036">
        <!--  <negotiator type="challenge" description="d:/hsqldb/users.properties"/>-->
    </acceptor>

    <repository name="repoCapella">
        <property name="overrideUUID" value="repoCapella"/>
        <property name="supportingAudits" value="true"/>
        <property name="supportingBranches" value="false"/>
        <property name="verifyingRevisions" value="false"/>
        <property name="currentLRUCapacity" value="10000000"/>
        <property name="revisedLRUCapacity" value="10000000"/>
        <property name="supportingEcore" value="true"/>

        <!-- Type of access control : userManager or securityManager -->
        <userManager type="auth" description="usermanager.properties" /> -->
        <!-- <securityManager type="collab" realmPath="securitymanager.properties"/> -->

        <store type="db">
            <mappingStrategy type="horizontalAuditingWithRanges">
                <property name="toManyReferences" value="ONE_TABLE_PER_REFERENCE"/>
                <property name="toOneReferences" value="LIKE_ATTRIBUTES"/>
                <property name="qualifiedNames" value="true"/>
                <property name="forceZeroBasedIndex" value="true"/>
                <property name="withRanges" value="true"/>
            </mappingStrategy>
            <jdbcDelegate type="preparedStatement">
                <!--
                    to explicitly force prepared statement caching (e.g., if statement
                    pooling is not supported by the JDBC driver, use <property
                    name="cacheStatements" value="enabled" /> Else, the implicit
                    default is: <property name="cacheStatements" value="guess" /> Which
                    guesses the correct setting based on the JDBC driver's metadata.
                    Also supported is the third setting "disabled".
                -->
            </jdbcDelegate>

            <dbAdapter name="h2-capella"/>
            <!--
                to use CLOB instead of VARCHAR to store description and documentation field : use the dbAdapter h2-capella instead of h2
                dbAdapter name="h2-capella"
            -->
            <dataSource class="org.h2.jdbcx.JdbcDataSource" uRL="jdbc:h2:db/h2/capella;LOG=1;CACHE_SIZE=65536;LOCK_MODE=0;UNDO_LOG=0" user="sa"/>
        </store>
    </repository>
```

Highlighted elements can be changed to customize the Team for Capella Server.

Note that many repository configuration options can not be changed anymore after the repository has been started the first time or if some data have been exported once to the server. If you need to change something in this configuration afterwards, you should then first delete the database files (files with db extension). A typical example is changing the name of the repository. The only elements you can change in the configuration file afterwards are *Type of access control : userManager, securityManager, ldap or none* and the *acceptor*.

Authenticated Configuration

To activate the authenticated server you have to set the line below in the **cdo-server.xml** file before the `<store >` tag. `<userManager type="auth" description="usermanager-config.properties"/>`

`usermanager.properties` is a path to the authenticated server configuration file. The path can be absolute or relative to the cdo-server.xml file.

```
users.file.path=users.properties
# ldap configuration
auth.type=ldap
auth.ldap.url=ldap://127.0.0.1:10389
auth.ldap.dn.pattern=cn={user},ou=people,o=sevenSeas
auth.ldap.filter=
```

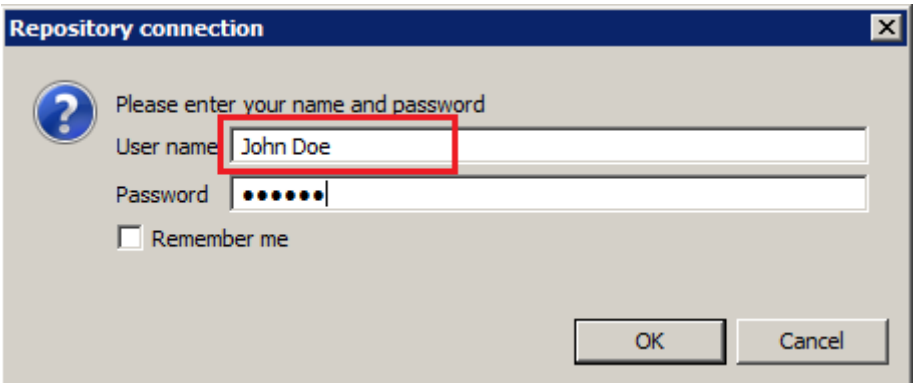
```
auth.ldap.tls.enabled=false
auth.ldap.truststore.path=
auth.ldap.truststore.passphrase=
# openID Connect configuration
#auth.type=openidconnect
#auth.openIDConnect.discoveryURL=https://login.microsoftonline.com/{tenant}/v2.0/.well
-known/openid-configuration
#auth.openIDConnect.tenant=organizations
#auth.openIDConnect.clientID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
#auth.openIDConnect.technicalUsers.file.path=technicalUsers.properties
#auth.openIDConnect.userInfoMatchClaim=name
```

- `users.file.path` is the name of the file containing the users. This file has to be copied into the root server installation folder. You can add new users by modifying the `users.properties` file.
- `auth.xxx` corresponds to the LDAP configuration or the OpenID Connect configuration. The properties are prefixed by **auth**. Beware to **uncomment at most the LDAP or the OpenID Connect configuration**.

The file `users.properties` contains entries which keys are the logins and values are the passwords. Note that space must be escaped with \ else it will be considered as a key-value separator. Examples:

```
admin=admin
John\ Doe=secret
```

This is the default mode, when Team for Capella is installed the server is set with a file authentication configuration.
You must not escape spaces in the login field required to connect to remote model (see Connect to remote model section).
The same applies when you create a new user through the **"security model"** (see User Creation section).



As access control modes are exclusive, other modes must be commented in the **cdo-server.xml** file:

```
<!-- <securityManager type="collab" .../> -->
<!-- <authenticator type="ldap" .../> -->
```

The server must be restarted to take into account the modifications done in the **cdo-server.xml** file.

On Client side, use the **User Management** view available in all Team for Capella clients. When using this view, the server does not need to be restarted after changes in the user accounts

User Profiles Configuration

To activate the user profile server you have to set the line below in the **cdo-server.xml** file before the `<store >` tag. The user profiles model is created at the first server launch.
Once activated, you must see this during the Team for Capella Server starting:

```
!ENTRY org.eclipse.emf.cdo.server.security 1 0 2016-03-24 11:32:27.413
!MESSAGE Security realm created in repoCapella.security
[INFO] Security extension started
```

```
<securityManager type="collab" realmPath="userprofile-config.properties" />
```

`userprofile-config.properties` is a path to the user profile configuration file. The path can be absolute or relative to the cdo-server.xml file.

```
realm.users.path=users.userprofile
administrators.file.path=administrator.properties
# ldap configuration
auth.type=ldap
auth.ldap.url=ldap://127.0.0.1:10389
auth.ldap.dn.pattern=cn={user},ou=people,o=sevenSeas
auth.ldap.filter=
auth.ldap.tls.enabled=false
auth.ldap.truststore.path=
auth.ldap.truststore.passphrase=
# openID Connect configuration
#auth.type=openidconnect
#auth.openIDConnect.discoveryURL=https://login.microsoftonline.com/{tenant}/v2.0/.well
-known/openid-configuration
#auth.openIDConnect.tenant=organizations
#auth.openIDConnect.clientID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
#auth.openIDConnect.technicalUsers.file.path=technicalUsers.properties
#auth.openIDConnect.userInfoMatchClaim=name
```

- `realm.users.path` is the name of the resource that contains the user profile model.
- `administrators.file.path` is a path to the administrators file. The path can be absolute or relative to the cdo-server.xml file. This file is only used to **initialize administrators** in the user profile model during the first start of the repository with the User Profile mode enabled (repository creation for example). It is mandatory because the definition of the user profile can only be done by an administrator.
- `auth.xxx` corresponds to the LDAP configuration or the OpenID Connect configuration. The properties are prefixed by **auth**. Beware to **uncomment at most the LDAP or the OpenID Connect configuration**.

Be aware that once the server has been launched with the User Profile mode enabled, modifications on this file will have no effect. If you want to manage the list of administrators, please have a look at User Pofiles documentation and especially at the Promote a User to Super User section if you want to promote an existing user to administrator. On the other hand, you can also make backups (shared projects and User Profiles model), stop the server, delete the database, modify the administrators files, restart the server and re-export your data.

As access control modes are exclusive, other modes must be commented in the **cdo-server.xml** file:

```
<!-- <userManager type="auth" .../> -->
<!-- <authenticator type="ldap" .../> -->
```

The server must be restarted to take into account the modifications done in the **cdo-server.xml** file.

Not Authenticated Configuration

This configuration allows to work with a CDO server without authenticating from a client.

Just comment **securityManager**, **userManager** and **authenticator** tags in the cdo-server.xml file:

```
<!-- <securityManager type="collab" .../> -->
<!-- <userManager type="auth" .../> -->
<!-- <authenticator type="ldap" .../> -->
```

The server must be restarted to take into account the modifications done in the **cdo-server.xml** file.

Activate LDAP authentication

Activate LDAP authenticator

You can activate LDAP authentication in three different ways:

- As an authenticator, the user must only be declared in the LDAP directory.
- In combination with authenticated server. In that case, the authentication requires that the user is declared in the user file and is authenticated with the LDAP directory.
- In combination with user profile server. In that case, the authentication requires that the user is defined in the user profile model and is authenticated with the LDAP directory.

The server must be restarted to take into account the modifications done in the **cdo-server.xml** file.

These ways are excluding themselves.

To activate LDAP authentication, as exclusive authenticator, the following authenticator tag must be added to the repository configuration in **cdo-server.xml**.

```
<authenticator type="ldap" description="ldap-config.properties" />
```

`ldap-config.properties` is a path to a properties file containing the LDAP authenticator configuration. This path may be relative to the CDO server configuration file or absolute.

As access control modes are exclusive, other modes must be commented in the **cdo-server.xml** file:

```
<!-- <userManager type="auth" .../> -->
<!-- <securityManager type="collab" .../> -->
```

Configure LDAP authenticator

The LDAP authenticator's configuration file is a properties file whose content could look like the following one:

```
ldap.url=ldap://127.0.0.1:10389
#ldap.url=ldaps://127.0.0.1:10389
ldap.dn.pattern=cn={user},ou=people,o=sevenSeas
ldap.filter=
ldap.tls.enabled=true
ldap.truststore.path=trusted.ks
ldap.truststore.passphrase=secret
```

where :

- **ldap.url** is the URL of the LDAP server, typically in the form `<ldap or ldaps>://<IP_address or domain_name>:<port>`
- **ldap.dn.pattern** is the pattern to define the LDAP query used to bind a user. It must contain a `{user}` part which will be replaced with the login provided by the user.
- **ldap.filter** is the LDAP query used to filter users by checking some attributes (optional). Different patterns are available to define this filter. For instance with the Apache DS sample (To download it, you can save the target of this link), to grant access to users having an email, the ldap filter pattern would be: *mail=**. As another example, to filter user (from a directory named «Users») members of a group named «grp1» in the domain «MyCompany.com» the filter to declare will be : *memberOf=CN=grp1,CN=Users,DC=MyCompany,DC=com*.
- **ldap.tls.enabled** is used for TLS enabling : *true* to enable TLS, *false* otherwise (non-SSL mode or use of deprecated LDAPS protocol). The default value is *true*.
- **ldap.truststore.path** is the absolute path to a certificate truststore (useful for self-signed certificates)
- **ldap.truststore.passphrase** is the truststore's passphrase (useful for self-signed certificates)

When the LDAP authenticator is used in User Profile or Authenticated configurations, those properties property keys must be prefixed by the **auth.** and the **auth.type=ldap** is needed to activate the LDAP authentification.

Unlike the other two configuration ways (with «user profile server» and «authenticated server»), in the «exclusive authenticator configuration», the properties are not prefixed by **auth**.

If the LDAP certificate has been signed by an official Certificate Authority it is not required to set the trust store path as the JVM already trusts the CA.

If you need to generate a self-signed certificate or need to create a trust store from an existing certificate please refer to the following section.

Use a self-signed or non CA-authentified certificate

In case the certificate is self-signed or the CA used in your certificate is not managed by the jvm, you will need to generate a truststore and reference this truststore from the configuration file.

Follow the Export and TrustStore creation steps to create the trust store.

Example of LDAP using multiple patterns

```
# ldap configuration
ldap.url=ldap://ldap.myCompany.com:389
ldap.dn.pattern=uid={user},ou=group1,o=orga1,dc=mycompany,dc=fr
ldap.dn.pattern.1=uid={user},ou=group2,o=orga1,dc=mycompany,dc=fr
ldap.dn.pattern.2=uid={user},ou=groupN,o=orgaM,dc=mycompany,dc=fr

ldap.tls.enabled=false
```

Configure LDAP with a manager

Some LDAP does not support anonymous binding (if your LDAP server doesn't even allow a query without authentication), then Capella would have to first authenticate itself against the LDAP server, and Capella does that by sending the «manager» DN and password. Using this specific connection, the user credentials (given by the user in the authentication popup) can be looked for in the LDAP tree.

This manager credentials needs to be provided in the properties file as it will not be asked to the user. These credentials are provided with the following properties:

- **ldap.manager.dn** : The login of the manager.
- **ldap.manager.password** : The password of the manager.

The search for the user himself in the LDAP is provided with the following properties:

- **ldap.user.search.base** : search pattern working like the **ldap.dn.pattern** field.
- **ldap.user.search.filter** : search filter working like the **ldap.filter** filed.

Example of LDAP configuration with a manager

```
# ldap configuration
ldap.url=ldap://ldap.myCompany.com:389
ldap.user.search.base=dc=myCompany,dc=com
ldap.user.search.filter=(&amp;(objectClass=account)(cn={user}))

# The manager credentials are useful for LDAP requiring authentication to run search
filters
ldap.manager.dn=uid=manager,ou=People,dc=myCompany,dc=com
ldap.manager.password=DerfOcDoocs6

ldap.tls.enabled=false
```

Configure LDAP with Active Directory

An LDAP using Active Directory provides a field `sAMAccountName` that is usually used as a key (like the "cn" field). Users can be identified using this field associated with a domain name after an "@" as separator. This leads to this pattern: sAMAccountName@DomainName. As the user identifies himself by providing only his identifier, not the domain name, the corresponding pattern is: {user}@DomainName.

Example of LDAP configuration with Active Directory

For instance, if the domain name is "MyCompanyDomain" then the LDAP pattern will be: `ldap.dn.pattern={user}@MyCompanyDomain`

```
# ldap configuration
ldap.url=ldap://ldap.myCompany.com:389
ldap.dn.pattern={user}@MyCompanyDomain

ldap.tls.enabled=false
```

Example of LDAP configuration with a manager and Active Directory

```
# ldap configuration
ldap.url=ldap://ldap.myCompany.com:389
ldap.user.search.base=dc=myCompany,dc=com
ldap.user.search.filter=(&amp;(objectClass=organizationalPerson)(name={user}))

# The manager credentials are useful for LDAP requiring authentication to run search
filters
ldap.manager.dn=manager@myCompany.com
ldap.manager.password=managerPassword

ldap.tls.enabled=false
```

Additional information

Sometimes, the structure of an Active Directory is not trivial and the exact path of a user in the Active Directory tree is not obvious. Here are some usefull command to get information about the Active Directory structure to reach the user "testUser" (testUser@myCompany.com)

```
dsquery * -filter "samaccountname=testUser" -attr *
```

```
dsquery group -name "groupname"
```

```
dsquery * -filter "samaccountname=testGroup" -attr *
dsquery * -filter "cn=testGroup" -attr *
```

How to complete the filter with the group

```
(& (objectClass=organizationalPerson)(sAMAccountName={user})
(memberOf=CN=grp1,CN=Users,DC=MyCompany,DC=com))
```

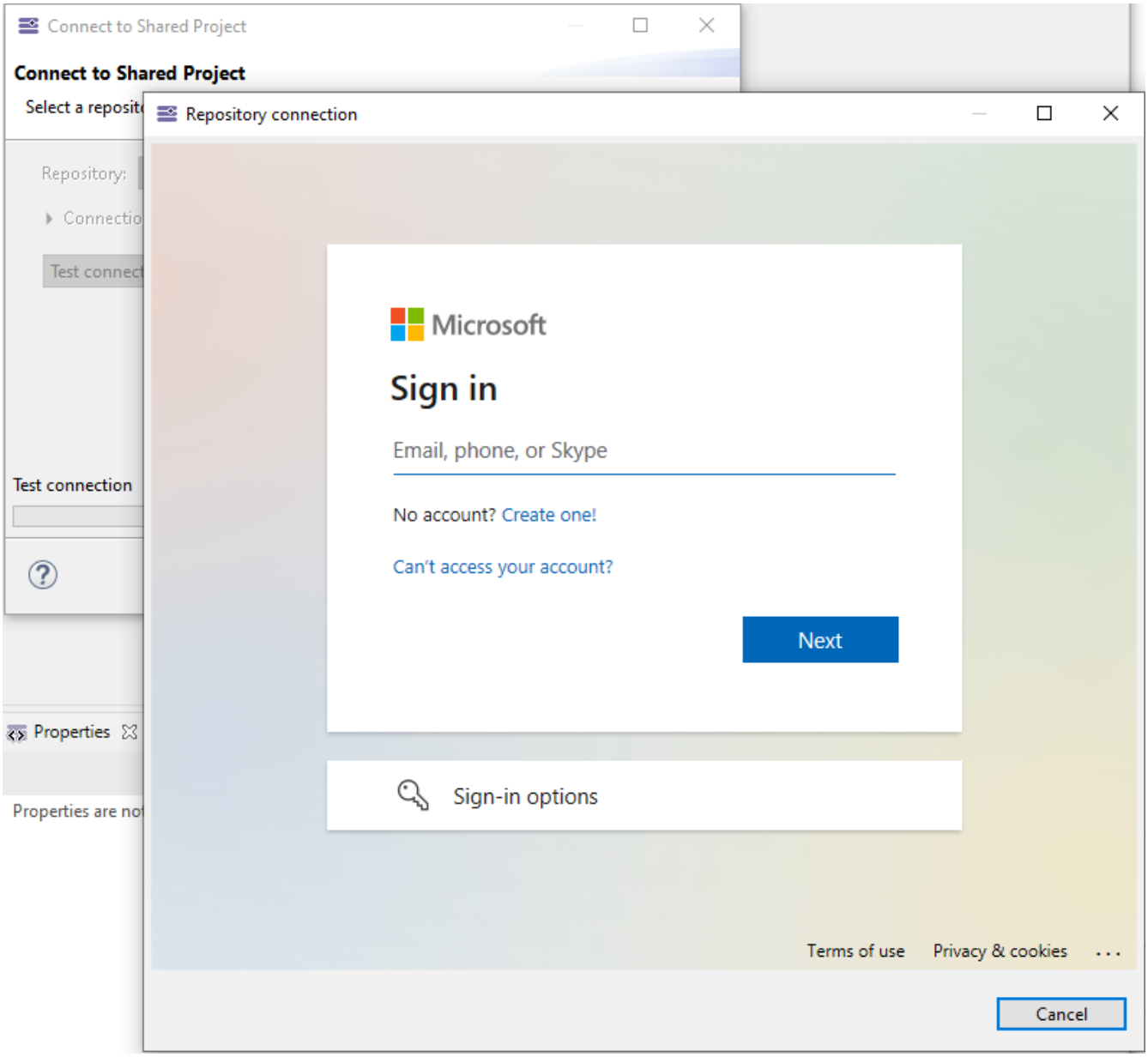
Usefull links:

- https://learn.microsoft.com/en-us/windows/win32/ad/naming-properties
- Use Powershell to get the information https://learn.microsoft.com/en-us/powershell/module/activedirectory/get-aduser?view=windowsserver2022-ps

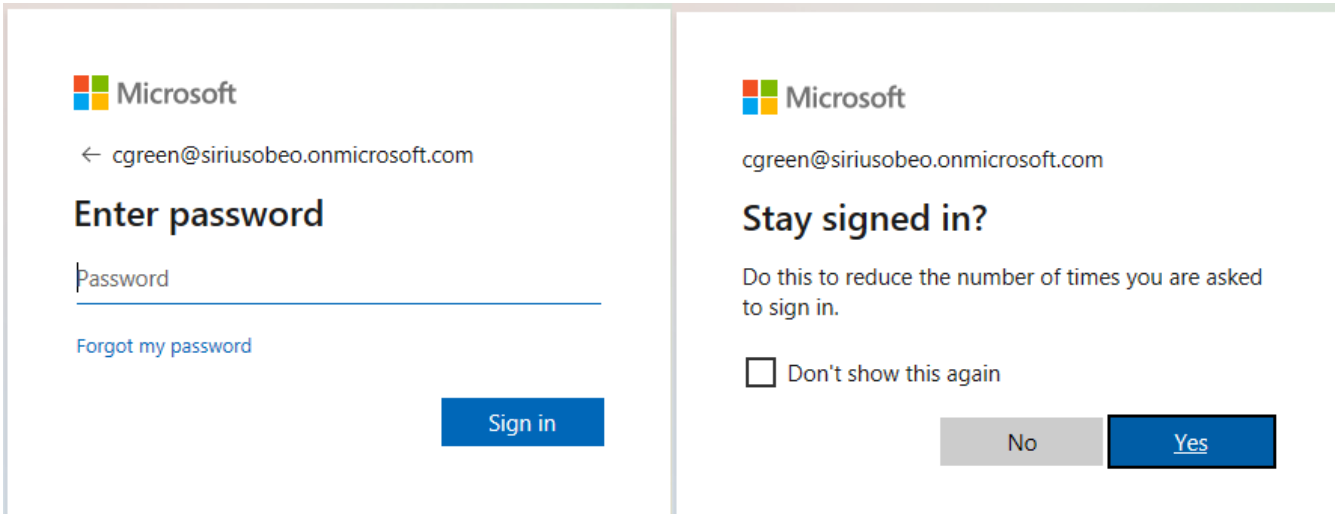
Activate OpenID Connect authentication

With a server set with an OpenID Connect Connect authentication, the user will be able to authenticate using the UI provided by the OpenID Connect Platform. Instead of having the default dialog where the user enters his login password, here the embedded T4C web server will display a popup web browser interacting with the OpenID Connect platform.

For instance, for a server set with Microsoft Entra ID, here is the user experience when the user clicks on the «Test Connection» button of the Connection wizard. A web browser is displayed and present a Sign-in interface provided by Microsoft Entra ID.

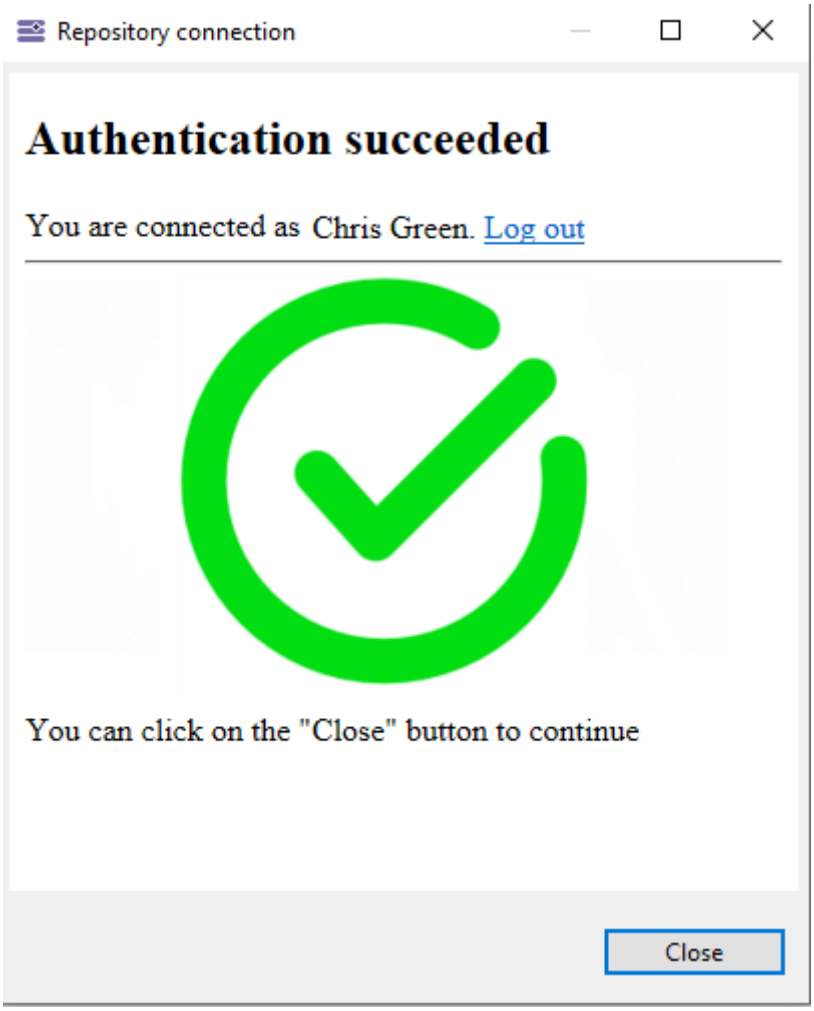


Then, the user follows the authentication process through the different web pages provided by the OpenID Connect platform depending on how it is configured.



Finally, the user will be presented a web page displaying if the authentication was successful or not. The user can close the browser and continue as usual. In this page, a «Logout» hyperlink allows to

logout the current user. The end-user is redirected to the sign-in page and may sign-in with another login.



Team for Capella authentication delegation trough OpenID Connect is currently supported towards Microsoft Entra ID and Keycloack.

In the next sections, configuration is shown with Microsoft Entra ID concepts.

Technical views such as CDO views or Administration views still authenticate with basic login/password credentials. See Configure OpenID Connect authenticator to know how to configure this credentials.

Configure Team for Capella server

Activate OpenID Connect authenticator

You can activate the OpenID Connect authentication:

- As an authenticator, the user must only be declared on the OpenID Connect platform.
- In combination with user profile server. In that case, the authentication requires that the user is defined in the user profile model and is authenticated with the OpenID Connect server.
- In combination with authenticated server. In that case, the authentication requires that the user is declared in the user file and is authenticated with the OpenID Connect server.

For the combination with both «user profile server» and «authenticated server», the user name to configure in Team For Capella must correspond to the attribute **"Name"** of the user in the OpenID Connect authentication platform.

The server must be restarted to take into account the modifications done in the **cdo-server.xml** file.

To activate the OpenID Connect authentication, as exclusive authenticator, the following authenticator tag must be added to the repository configuration in **cdo-server.xml**. Make sure the other tags are commented.

```
<authenticator type="openidconnect" description="openid-config.properties" />
```

`openid-config.properties` is a path to a properties file containing the OpenID Connect authenticator configuration. This path may be relative to the CDO server configuration file or absolute.

As access control modes are exclusive, other modes must be commented in the **cdo-server.xml** file:

```
<!-- <userManager type="auth" .../> -->
<!-- <securityManager type="collab" .../> -->
```

Finally, the OpenID Connect authentication requires a web server in order to securely communicate with the OpenID Connect platform. If the CDO server is configured with the OpenID Connect authentication mode, then it will require to activate the embedded web server for this secure communication.

Configure OpenID Connect authenticator

`<installation folder>/server/configuration/openid-config.properties` is the OpenID Connect authenticator's configuration file. It is a properties file whose content could look like the following one:

```
openIDConnect.discoveryURL=https://login.microsoftonline.com/{tenant}/v2.0/.well-
known/openid-configuration
openIDConnect.tenant=organizations
openIDConnect.clientID=79bce8de-7542-4b90-bf18-XXXXXXXXXXXX
openIDConnect.technicalUsers.file.path=technicalUsers.properties
openIDConnect.userInfoMatchClaim=name
```

where :

- **openIDConnect.discoveryURL** is the URL of the OpenID Connect metadata document (RFC) that contains the information required for the authentication. If it contains `{tenant}`, it will be replaced with the value of the **openIDConnect.tenant** property.
- **openIDConnect.tenant** controls the type of user profile that will be able to authenticate. For Microsoft Entra ID, it corresponds to the Directory (tenant) ID of your App Registration.
- **openIDConnect.clientID** is the application ID that the OpenID Connect platform assigned to

your application.

- **openIDConnect.technicalUsers.file.path** is a relative path to a properties file that contains credentials(login, password) used for technical views such as CDO views or Administration views.
- **openIDConnect.userInfoMatchClaim** is the attribute used by the User Management and User Profile access control modes to identify a user. This field defines the type of value to specify in user configuration files. It is optional, the value is set to *name* by default.

It is also possible to specify the values of the `&responseType`, `&scope`, `&state` parameters used in the OpenID Connect authentication request. These parameters can be configured by modifying the server VM arguments with:

- **-Dfr.obeo.dsl.viewpoint.collab.server.openid.responseType=<responseType>**
- **-Dfr.obeo.dsl.viewpoint.collab.server.openid.scope=<scope>**
- **-Dfr.obeo.dsl.viewpoint.collab.server.openid.state=<state>**

If these parameters are not specified, the default parameters will be used`: `&responseType=id_token+token`, `&scope=openid+profile+email`, and random UUID will be used as value for the `&state` clause added to the request.

If additional specific elements are required (e.g. `&acr_value=xxxXXX`), it can be passed with **-Dfr.obeo.dsl.viewpoint.collab.server.openid.complement=<complement>** (including the `&`), it will be appended at the end of the request.

Three additional *experimental* properties allow to customize the way the server will handle authentication response:

- user information retrieval (to display it in the success page, collaborative session details and decorators):
  - **-Dfr.obeo.dsl.viewpoint.collab.server.openid.userInfoPayload=<userInfoEndpoint|access_token|id_token>** (default value if not set: **userInfoEndpoint**)
- repository access check (to allow access to a repository if the chosen payload value contains the repository name):
  - **-Dfr.obeo.dsl.viewpoint.collab.server.openid.repositoryFilterPayload=<userInfoEndpoint|access_token|id_token>** (default value if not set: no filtering)
  - **-Dfr.obeo.dsl.viewpoint.collab.server.openid.repositoryFilterMatchClaim=<claimName>** (default value if not set: **t4c_repository**)

Configure embedded web server for OpenID Connect authentication

As presented before, the OpenID Connect Authentication requires a web server in order to authenticate securely. This is the same web server as the one providing the web services (REST API)

for repository management. See in the dedicated section how to install and activate this experimental feature.

To activate the OpenID Connect support, you need then to set the value of the `admin.server.jetty.auth.openidconnect.enabled` property to true in `<installation folder>/server/configuration/fr.obeo.dsl.viewpoint.collab.server.admin/admin-server.properties`.

Note that if you do not have the Team for Capella server and all the Team for Capella clients installed on the same machine, you will need to configure the web server in **https** mode. Indeed, this is a security required from the OpenID Connect platform. So,

- if the Team for Capella server is local to the Team for Capella client you may use `http` protocol with `localhost`. This is the default configuration.
- if the Team for Capella server is installed on a different machine than the Team for Capella client you must configure the admin server with `https`.

To configure the admin server with `https`, do the following changes in `<installation folder>/server/configuration/fr.obeo.dsl.viewpoint.collab.server.admin/admin-server.properties`

```
# Jetty configuration
admin.server.jetty.https.enabled=true

# The next three line will be needed if the '${admin.server.jetty.https.enabled}
option is set to true.'
admin.server.jetty.ssl.host=0.0.0.0
admin.server.jetty.ssl.port=8443
admin.server.jetty.ssl.keystore.path=${currentDir}/<keystoreFile>
admin.server.jetty.ssl.keystore.passphrase=<password>
```

- **admin.server.jetty.ssl.host** : let it as it is
- **admin.server.jetty.ssl.port** is the port to use for the **redirect url** in the configuration of the OpenID Connect authentication platform
- **admin.server.jetty.ssl.keystore.path** must lead to a keystore file. See Managing certificate to know how to do it.
- **admin.server.jetty.ssl.keystore.passphrase** is the password used for the keystore.

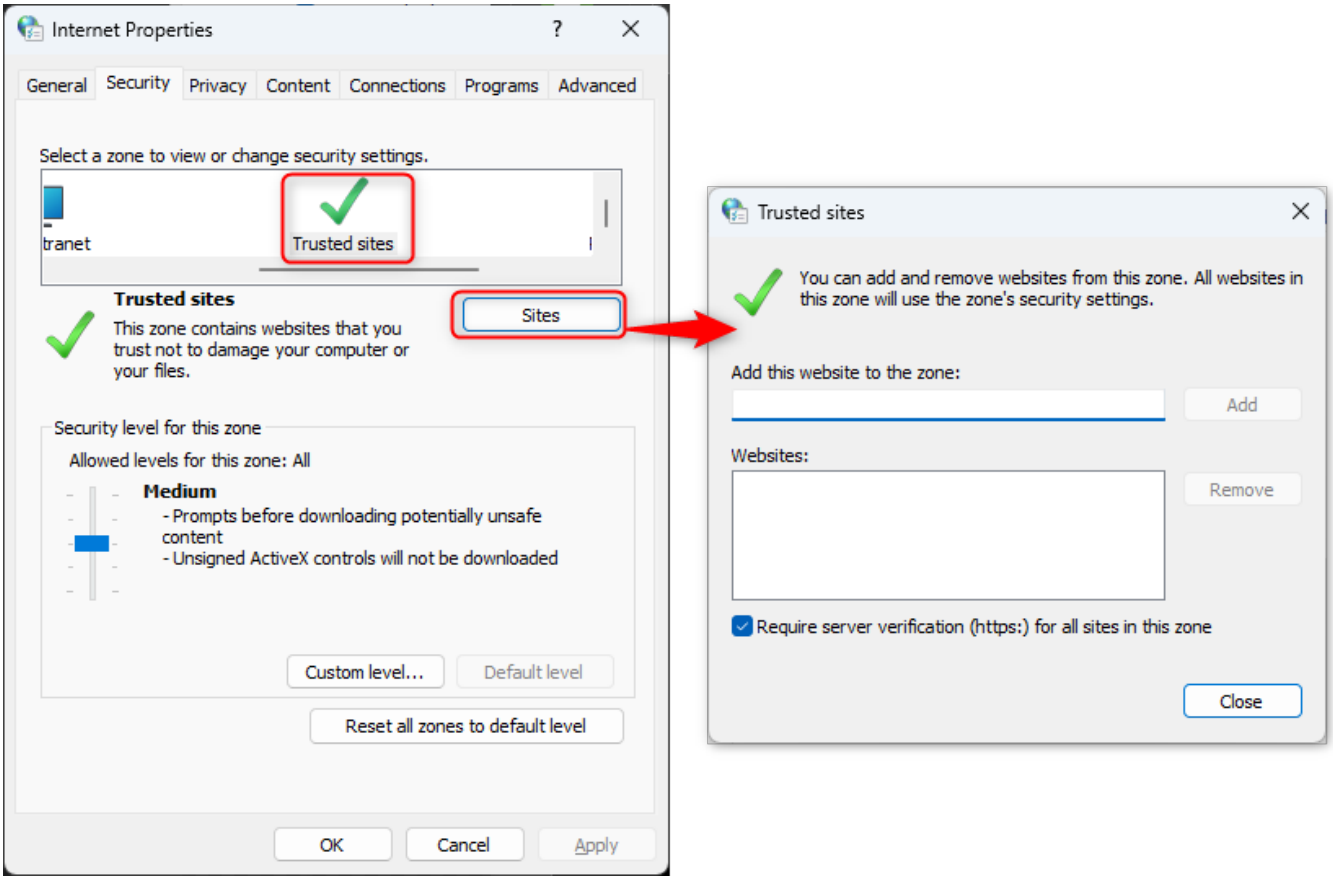
Configure the client for OpenIdConnect

Once the authentication with OpenIDConnect succeded the T4C client browser will call T4C admin server on redirect URI passing some information inside cookies. So The T4C client browser needs to trust the URL and authorize cookies.

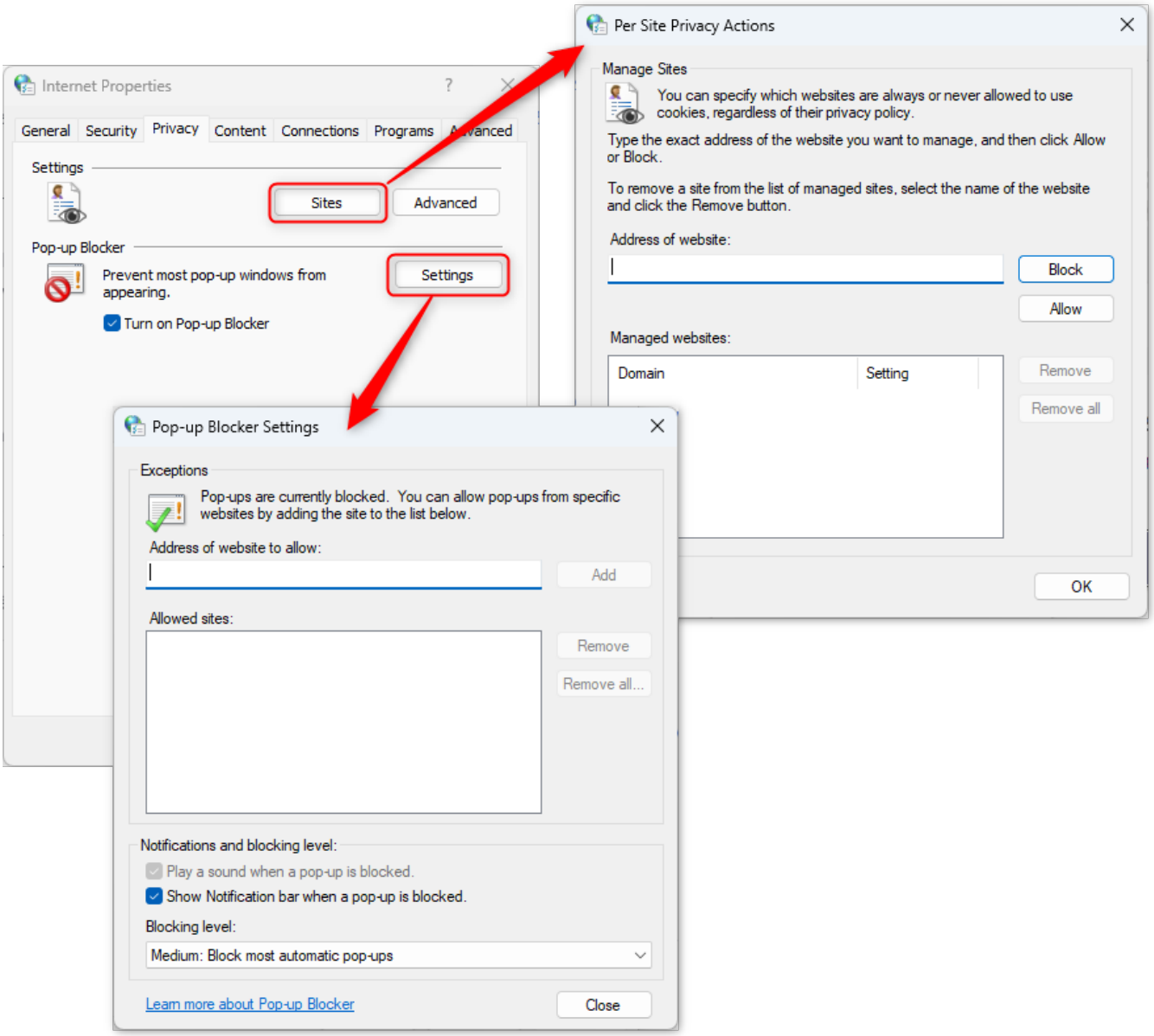
On windows, the configuration is done in `Internet properties`.

- Open `Internet properties`
- Select *Security* tab
- Select *Trusted sites* then *Sites* button

- Add the T4C admin server URL for example `https://yourDNS`



- Select *Privacy* tab
- Click *Sites* button
- Allow the T4C admin server URL for example `https://yourDNS`
- Click *Settings* button
- Allow the T4C admin server URL for example `https://yourDNS`



Configure the application on the OpenID Connect platform

On the OpenID Connect platform, there is one property that requires to be properly set: the **redirect URI**. Indeed, the embedded web server expects that the redirect URI is the page `/auth/redirect`. This means that the redirect URI must be set to

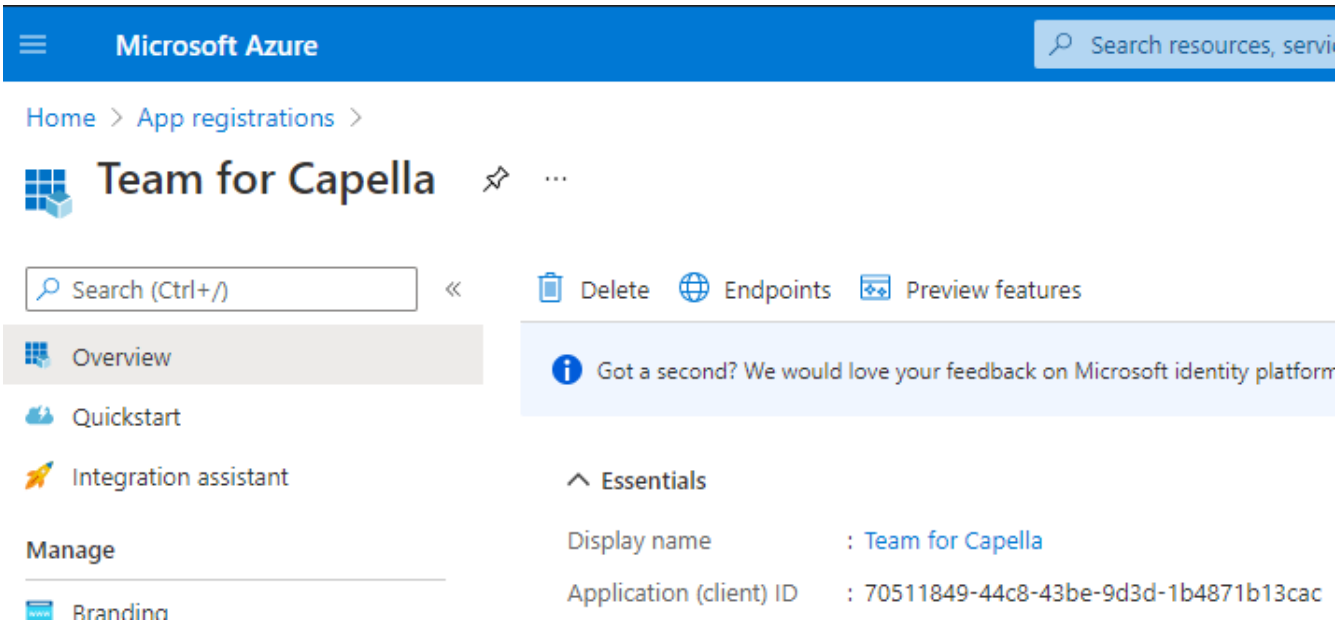
- either `http://localhost:8080/auth/redirect` if the Team for Capella server is local to the Team for Capella client
- or `https://<IP admin server>:8443/auth/redirect` if the Team for Capella server is installed on a different machine than the Team for Capella client.

Configure OpenID Connect authenticator with Microsoft Entra ID

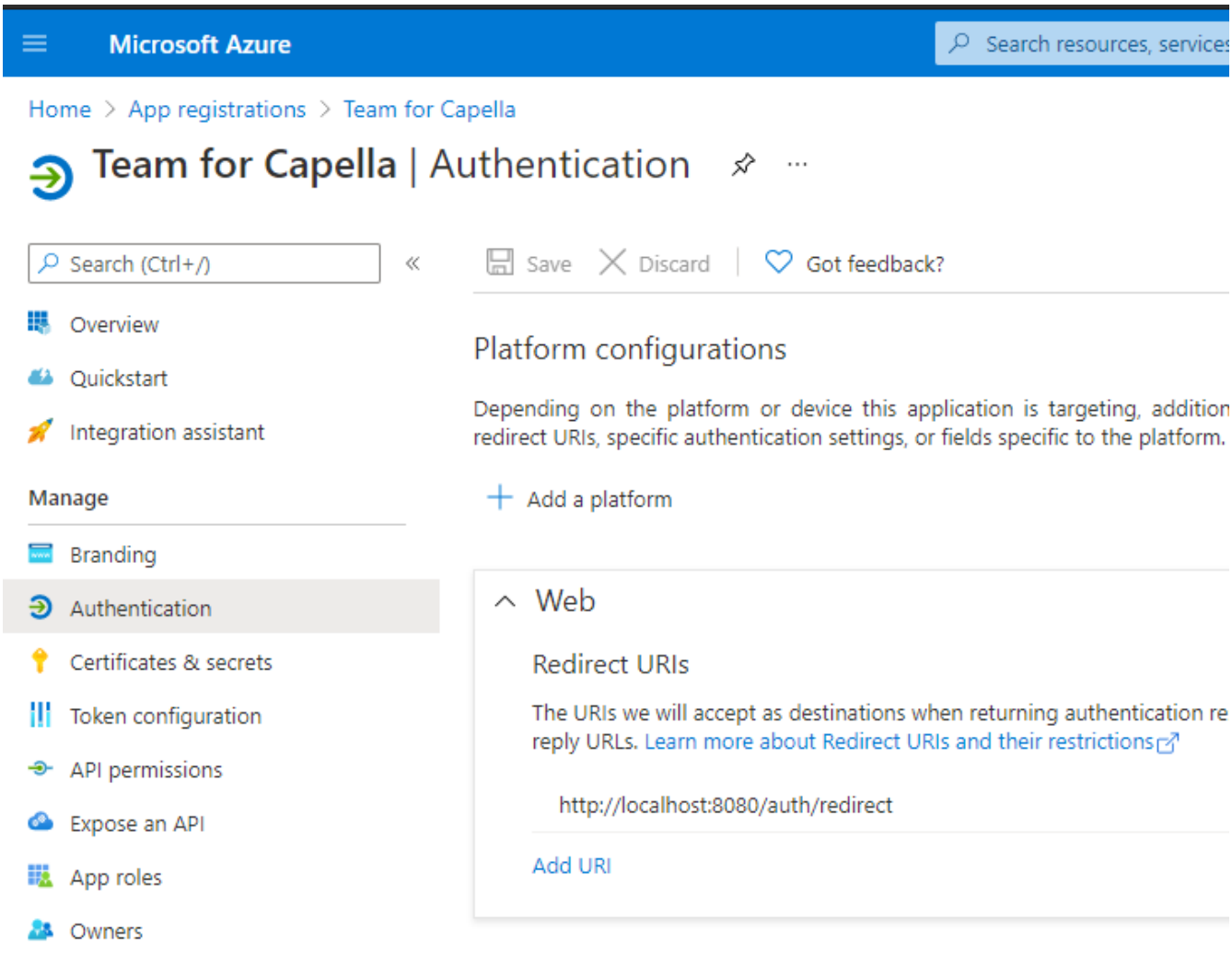
If your OpenID Connect platform is Microsoft Entra ID, here is a quick way to find how to configure the OpenID Connect authenticator in Team for Capella.

First, the `openIDConnect.discoveryURL` is provided by the OpenID Connect platform itself, not by your application. For Microsoft Entra ID, this protocol is presented in the online documentation. On the same page, there is a list of the different values the `openIDConnect.tenant`.

For the `openIDConnect.clientID`, you will need to look for it in the application you created in Microsoft Entra ID in order to use it for authentication from Team for Capella. From the Microsoft Entra ID home page, you can select **App registration**. Select your application for Team for Capella. From the overview, you can see the **Application ID**.

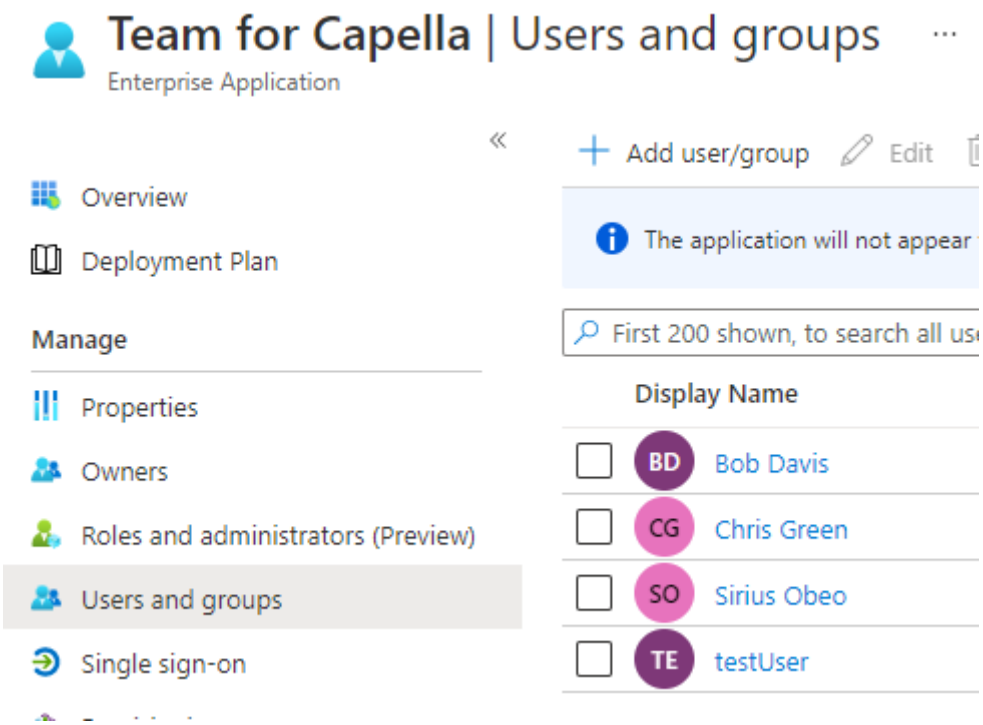


Note that from this menu, you must set the **redirect URI** from the menu **Authentication**. In Platform configuration add a **Web** platform and set the redirect URI.

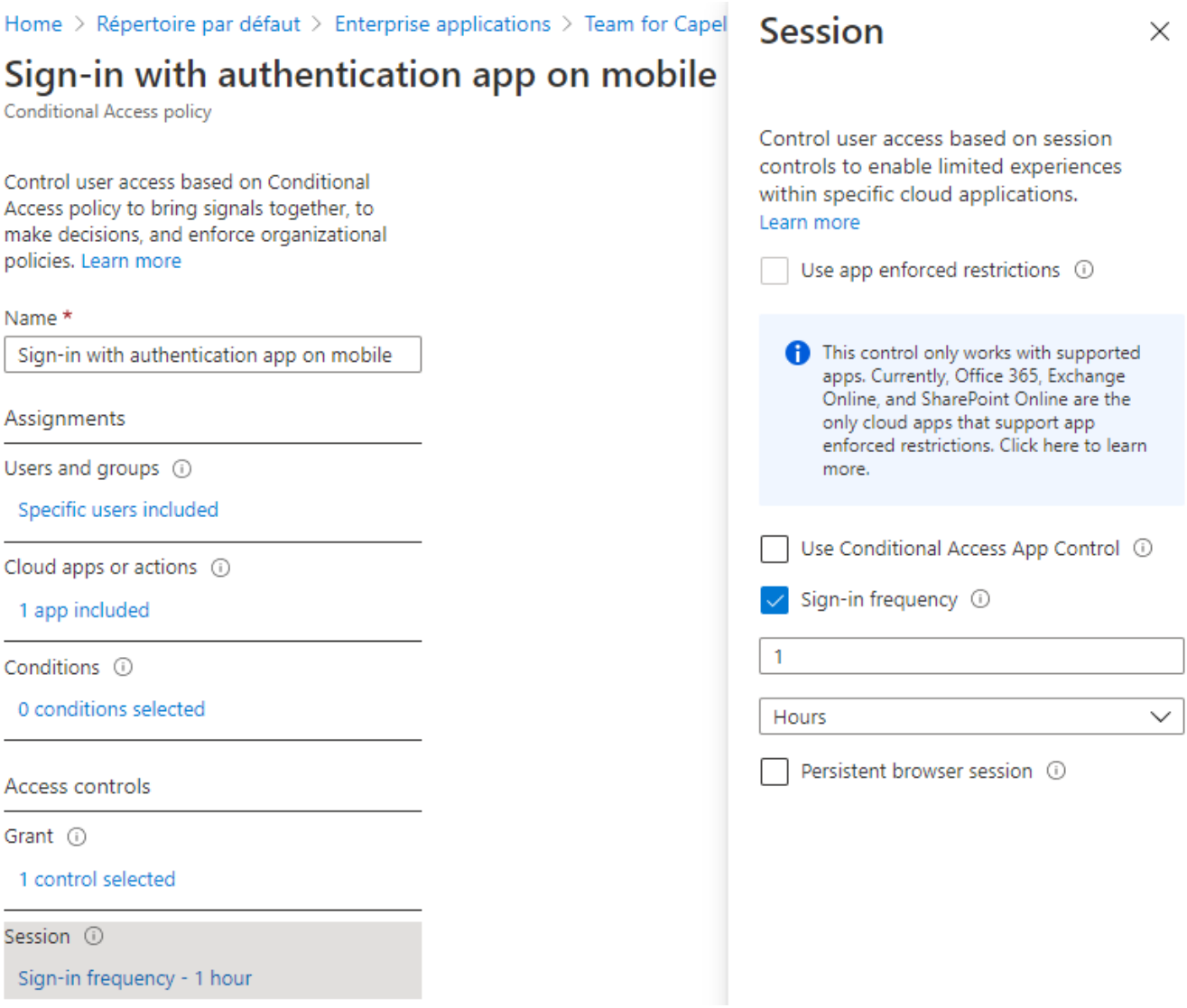


The last property, `openIDConnect.domainURL`, depends on the location/address of the web server and is not linked with the OpenID Connect configuration.

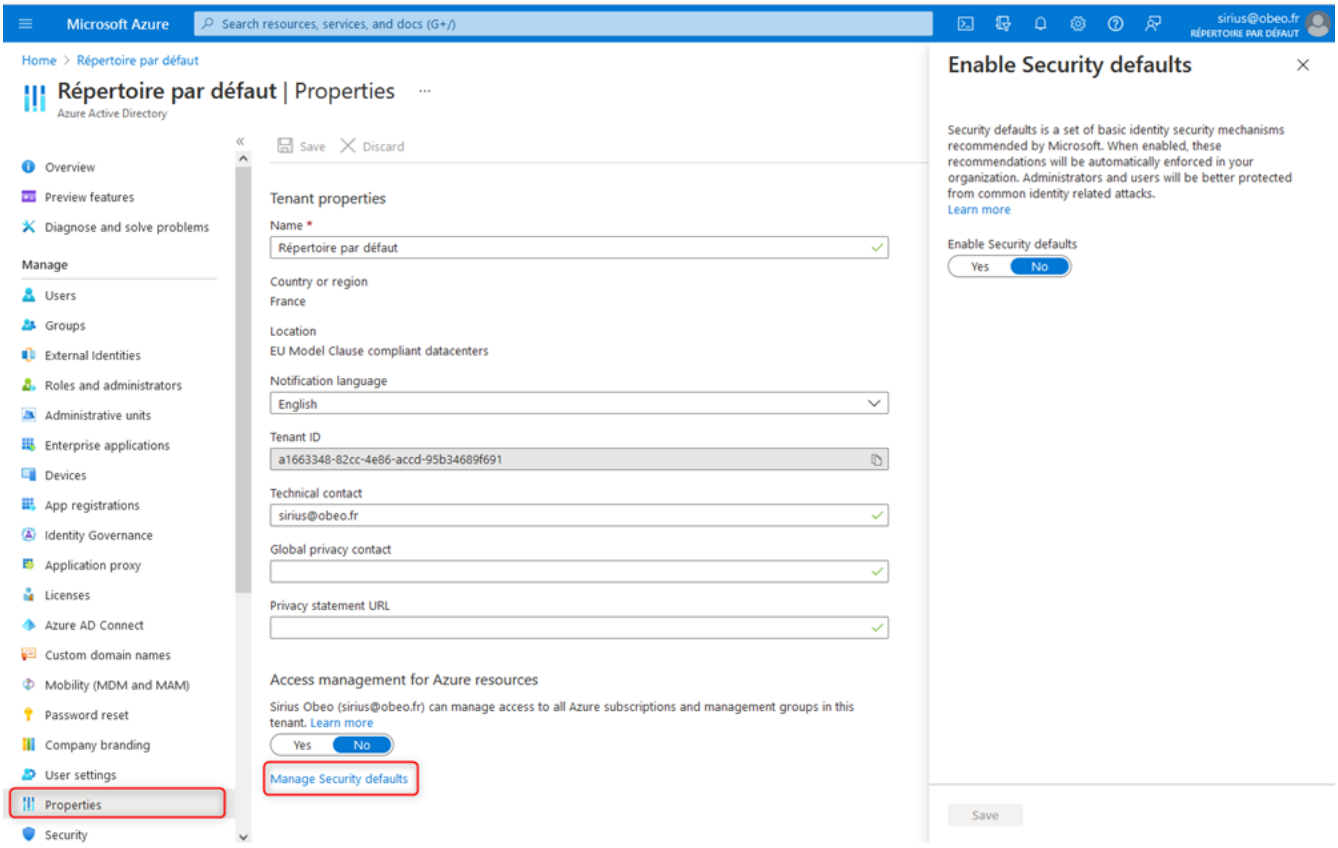
On your application, do not forget to add the users that will be able to authenticate to the application:



It is also recommended to create a conditional access policy (Security/Conditional Access) so you can set a timeout to the session once users are authenticated. You can also define how users are grant access (for instance with multi-factor authentication).

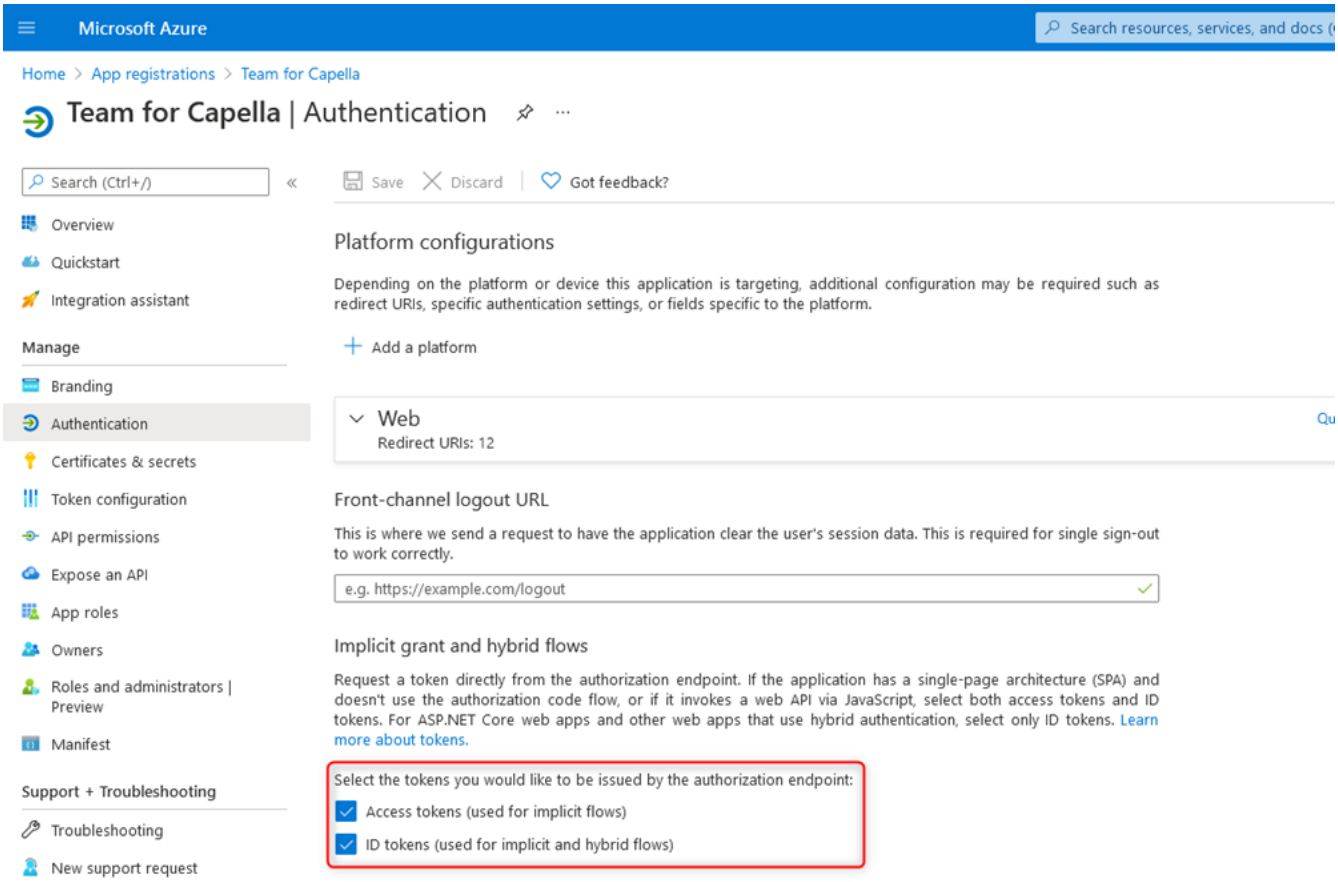


Note that to be able to add conditional access policies, you need to disable the security defaults.



Note that the following options must be activated because the authentication uses the implicit grant

- Access tokens (used for implicit flows)
- ID tokens (used for implicit and hybrid flows)



Audit mode

The *Audit mode* aims to configure the server so it keeps tracks of all versions of each object in the CDO Server database. It is required for comparing different versions of the model for example. There are two different auditing configurations: *Audit* and *Audit with ranges*.

This *Audit with ranges* mode has been the default mode between Team for Capella 1.3.0 and Team for Capella 5.0.0.

The *Audit* mode is the default mode since Team for Capelle 5.1.0 to improve user-side performances (export, export with override, semantic browser refresh, ...)

The difference between the two modes is in the storage of lists: when the *with ranges* variant is used, the database stores only the delta between each versions of lists. This implies to load all preceding revisions of a list to compute a given state. But for some situations, it can slow the growth of the database. An analysis of the project can lead to a recommendation to switch to this mode.

When using the auditing modes, the size of the database might need to be monitored. If the database size grows bigger than 4 GB, the user might need to clear it if he encounters performance issues. That is to say, importing the models from the server, clearing the database and then importing the models back in the new database. Be aware that after doing this operation he will not be able to compare new commits against the commits done before the clearance. Some benchmarks have been done, after 10 000 commits modifying semantic and graphical elements this size have never been reached. In this context, modification and saving model times increase slightly compared to a server that does not have audit mode enabled. However both operations still feel smooth for the user.

Be aware that it is not possible to switch between «Audit», «Audit with ranges» or "non «Audit» modes on a CDO server that holds models. The switch has to be done on a empty CDO server database.

In order to disable the Audit mode you have to change **cdo-server.xml** to:

- Set the audit mode property to *false* (default value is *true*).

```
<property name="supportingAudits" value="true"/>
```

- Change the mapping strategy to *horizontalNonAuditing* (default value is *horizontalAuditing*).

```
<mappingStrategy type="horizontalNonAuditing">
```

- Remove or comment the property *withRanges* in the mapping strategy properties.

```
<mappingStrategy type="horizontalNonAuditing">
    ...
    <!-- property name="withRanges" value="false"/ -->
</mappingStrategy>
```

In order to (re-)activate the Audit mode you have to change **cdo-server.xml** to:

- Set the audit mode property to *true* (default value is *true*).

```
<property name="supportingAudits" value="true"/>
```

- Change the mapping strategy to *horizontalAuditing* (default value is *horizontalAuditing*).

```
<mappingStrategy type="horizontalAuditing">
```

- Add the property *withRanges* in the mapping strategy properties.

```
<mappingStrategy type="horizontalAuditing">
    ...
    <property name="withRanges" value="false"/>
</mappingStrategy>
```

In order to activate the Audit *with ranges* mode you have to change **cdo-server.xml** to:

- Set the audit mode property to *true* (default value is *true*).

```
<property name="supportingAudits" value="true"/>
```

- Change the mapping strategy to *horizontalAuditingWithRanges* (default value is *horizontalAuditing*).

```
<mappingStrategy type="horizontalAuditing">
```

- Add or modify the property *withRanges* in the mapping strategy properties.

```
<mappingStrategy type="horizontalAuditing">
    ...
    <property name="withRanges" value="false"/>
</mappingStrategy>
```

Activate WebSocket connection

It is possible to activate a WebSocket connection between the client and the CDO server. Both client and server have to be configured accordingly.

Client configuration

On client side, users will have to use WS or WSS connection types regarding the configuration of the server.

The client side configuration will depend on the global deployment of the current server and the use of the WS and WSS connection types.

Then a user will have to use the following parameters to connect to the repository:

- Repository Host: localhost or ip/url of the server,
- Port Number: 8080 (value of `admin.server.jetty.port` or `admin.server.jetty.ssl.port` if HTTPS is enabled, or specific proxy port if Team for Capella is deployed behind a proxy)
- Repository Name: repoCapella
- Connection type: WS (WSS if Jetty has been deployed in HTTPS or is behind an HTTPS proxy)

When the REST Admin server runs in HTTPS mode, it will be configured with a certificate.

Add certificate in Capella client JRE

You need to add the public certificate into the JRE of the T4C client. The JRE is located in `<T4C client>/jre`

Assuming that the private certificate has been provided by the customer. The public certificate can be generated directly from the private.

You can use `keytool` to generate the public certificate or use `Keystore explorer` that can be download from `https://keystore-explorer.org/`

With `Keystore explorer`

- Open the `certificate.jks` used by the t4c server
- Select the certificate and export it. You get a `certificate.cer` file
- Open `Keystore explorer` and import `capella/jre/lib/security/cacert`
- Click on `import a trusted certificate` and select `certificate.cer`
- If the certificate is structured in tree with a root certificate import all certificates
- Do not forget to save

What if a Web Application Server(WAF) or a proxy stands between the T4C client and the T4C server

Context: the installation architecture is

- T4C client on client VM
- a WAF proxy that accept 443 connection
- a T4C server on server VM that accepts 8443

Issue when clicking `Test Connection` from the T4C client : `Unable to find valid certification path to requested target`

Analysis

The certificate is configured on the server. Is it the same certificate that is installed on the client JRE?

To check that

- From the client, open a Navigator and enter the address `https://t4cServerDNS:443`
- Open or export the certificate from the navigator url toolbar or from the navigator settings
- Check the certificate serial number
- From the server, open `Keystore explorer`. You can download it from `https://keystore-explorer.org/`
- Open the `certificate.jks` used by the t4c server
- Check the certificate serial number
- Expected : if serial numbers are not the same it is logical to have the issue

Pay attention to compare the proper serial number because sometimes the certificate used by the server contains a root certificate and other dedicated certificate

Solution

To solve the issue, we need to add the certificate used by the WAF in the client JRE `cacert`

- In the client VM, from the navigator, download the **root certificate** returned by the WAF using your navigator (directly from the url toolbar or using settings)
- Open `Keystore explorer` and import `capella/jre/lib/security/cacert`
- Click on `import a trusted certificate` and select the download certificates
- If the certificate is structured in tree with a root certificate import all certificates
- Do not forget to save

Restart capella and recheck the connectionSSL connection

Autosigned or untrusted certicate

If this certificate is self-signed or untrusted, the following system properties can be added in the client capella.ini file in order to configure the security checks:

- Trust any certificate (self-signed for example): `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.trustall=true`
- Endpoint Identification Algorithm: `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.endpointIdentificationAlgorithm`
- TrusStore passphrase: `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.passphrase`
- TrusStore URI: `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.trust`
- TrustStore type: `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.trust.type`
- Trust Manager Factory Algorithm: `-Dfr.obeo.dsl.viewpoint.collab.https.jetty.ssl.context.trust.manager.factory.algorithm`

Those properties are used to configure the Jetty's org.eclipse.jetty.util.ssl.SslContextFactory).

Additional properties might be needed, see server configuration section.

Tools configuration

When WebSocket transport is activated on the server, the importer and other tools must be configured accordingly to be successful. The same configuration than the client needs to be done in the -vmargs section of the tools script (importer.bat, maintenance.bat, exporter.bat, ...).

Server configuration

The REST Admin Server and the CDO Server need to be configured to enabled the Net4J WebSocket-based transport:

- the property `admin.server.jetty.net4j.enabled=true` in **<TeamForCapella installation folder>/server/configuration/fr.obeo.dsl.viewpoint.collab.server.admin/admin-server.properties** allows to deploy the Net4J Websocket servlet.
- a specific acceptor must be added in the **cdo-server.xml** with type `ws` or `wss`.
  - `<acceptor type="ws"/>` is the simplest and default WebSocket-based acceptor. Additional configurations are explained below.
  - It can be used to replace the default acceptor (`<acceptor type="tcp" listenAddr="0.0.0.0" port="2036"/>`) or as an additional one.

The move from a *Websocket-based* transport to a *SecuredWebsocket-based* transport can be done through Jetty configuration by enabling HTTPS, or with the use of an HTTPS reverse proxy server (Nginx or Apache for example).

Optional configuration

Here is a list of optional settings which will impact both server and clients configurations:

- change the acceptor name to be specific to your product: from `ws://1.2.3.4:8080/net4j/@default` to `ws://1.2.3.4:8080/net4j/@YourAcceptorName`
  - server: change the acceptor tag in the cdo-server.xml to be `<acceptor type="ws" name="YourAcceptorName" />`
  - client: `-Dfr.obeo.dsl.viewpoint.collab.net4j.ws.acceptor=YourAcceptorName` (same value than name attribute of the acceptor tag used on the server side).
- change the HTTP REST endpoint from the default value (`/net4j`) to the path of your choice: from `ws://1.2.3.4:8080/net4j/` to `ws://1.2.3.4:8080/your/path/`
  - server: set `admin.server.jetty.net4j.path=your/path`
  - client: `-Dfr.obeo.dsl.viewpoint.collab.net4j.ws.path=your/path`
- have basic authentication on the Net4J servlet (same credentials than the one used for the other servlets, see `openapi/`):
  - for tests purpose only: login passwords might be available from the installation details of the eclipse platform and in the executable companion .ini file.
  - server: `admin.server.jetty.net4j.remove.public.constraint=true`

- client:
  - declare the login: `-Dorg.eclipse.net4j.internal.ws.WSClientConnector.clientBasicAuth.login=sampleuser`
  - declare the password: `-Dorg.eclipse.net4j.internal.ws.WSClientConnector.clientBasicAuth.password=samplepassword`

Activate SSL connection

It is possible to activate a SSL connection between the client and the CDO server.

Both client and server have to be configured accordingly.

On server side a keystore has to be set/ up and, on client side, a trust store containing the key store public key has to be set up. See chapter Managing certificate to generate keystore and truststore.

Client configuration

Add the following lines in the client capella.ini file:

```
-Dorg.eclipse.net4j.tcp.ssl.passphrase=secret
-Dorg.eclipse.net4j.tcp.ssl.trust=file:///<trusted.ks absolute path>
```

Tools configuration

When SSL is activated on the server, the importer and other tools must be configured accordingly to be successful. Add the following lines in the script files (importer.bat, maintenance.bat, exporter.bat):

```
-Dorg.eclipse.net4j.tcp.ssl.passphrase=secret ^
-Dorg.eclipse.net4j.tcp.ssl.trust=file:///<trusted.ks absolute path> ^
```

Server configuration

In the **cdo-server.xml** configuration file the acceptor has to be configured to accept SSL connections `<acceptor type="ssl" listenAddr="0.0.0.0" port="2036"/>` Set the acceptor type to ssl.

Add the following lines in the server ini file:

```
-Dorg.eclipse.net4j.tcp.ssl.passphrase=secret
-Dorg.eclipse.net4j.tcp.ssl.key=file:///<server.ks absolute path>
```

Managing certificate

Keytool can be used to create and manage certificates and stores. This tool is provided with the JDK and its documentation is available here.

Generate a keystore

The keystore contains certificate information, private and public key. To generate it use the following command:

```
keytool -genkey -ext SAN=IP:<server IP> -keyalg "RSA" -dname o=sevenSeas -alias
keystore_alias -keystore server.ks -storepass secret -validity 730 -keysize 4096
```

-ext: For example, <server IP> may be the LDAP server for SSL connection between CDO server and LDAP server or may be the CDO Server for SSL connection between client and CDO server.

-dname: optional. It initializes the metadata of your organization.

Sign your certificate from a certificate authority(optional)

This step is optional and you may then proceed with Export certificate from a keystore. For that step, you have to give your certificate signature request(server.csr) to your certificate authority(CA) which, in return will provide a signed certificate(server.crt).

```
keytool -certreq  -alias keystore_alias -file server.csr -keystore "server.ks"
```

The two steps below allow to import root certificate and intermediary certificate.

```
keytool -import -alias Root_CA -keystore server.ks -file Root_CA.cer
keytool -import -alias Server_CA -keystore server.ks -file Server_CA.cer
```

Then, import the signed certicated into the server.ks keystore.

```
keytool -import -alias keystore-signed -keystore server.ks -file server.crt
```

Export certificate from a keystore

To export a certificate from an existing keystore the following command can be used :

```
keytool -export -keystore server.ks -alias keystore_alias -file server.cer
```

This command asks for the store's passphrase and then create a *server.cer* file containing the certificate previously created.

Create a truststore from a certificate

It is advised to not export the whole keystore on clients. Creating a truststore containing only the certificate and public key is recommended. This truststore is intended to be deployed on clients which need to connect to the server.

```
keytool -import -file server.cer -alias keystore_alias -keystore trusted.ks -storepass
secret
```

This command creates a new truststore in file *trusted.ks*. This truststore contains the server's public key, it can be copied on client machines and referenced via the *truststore.path* configuration key.

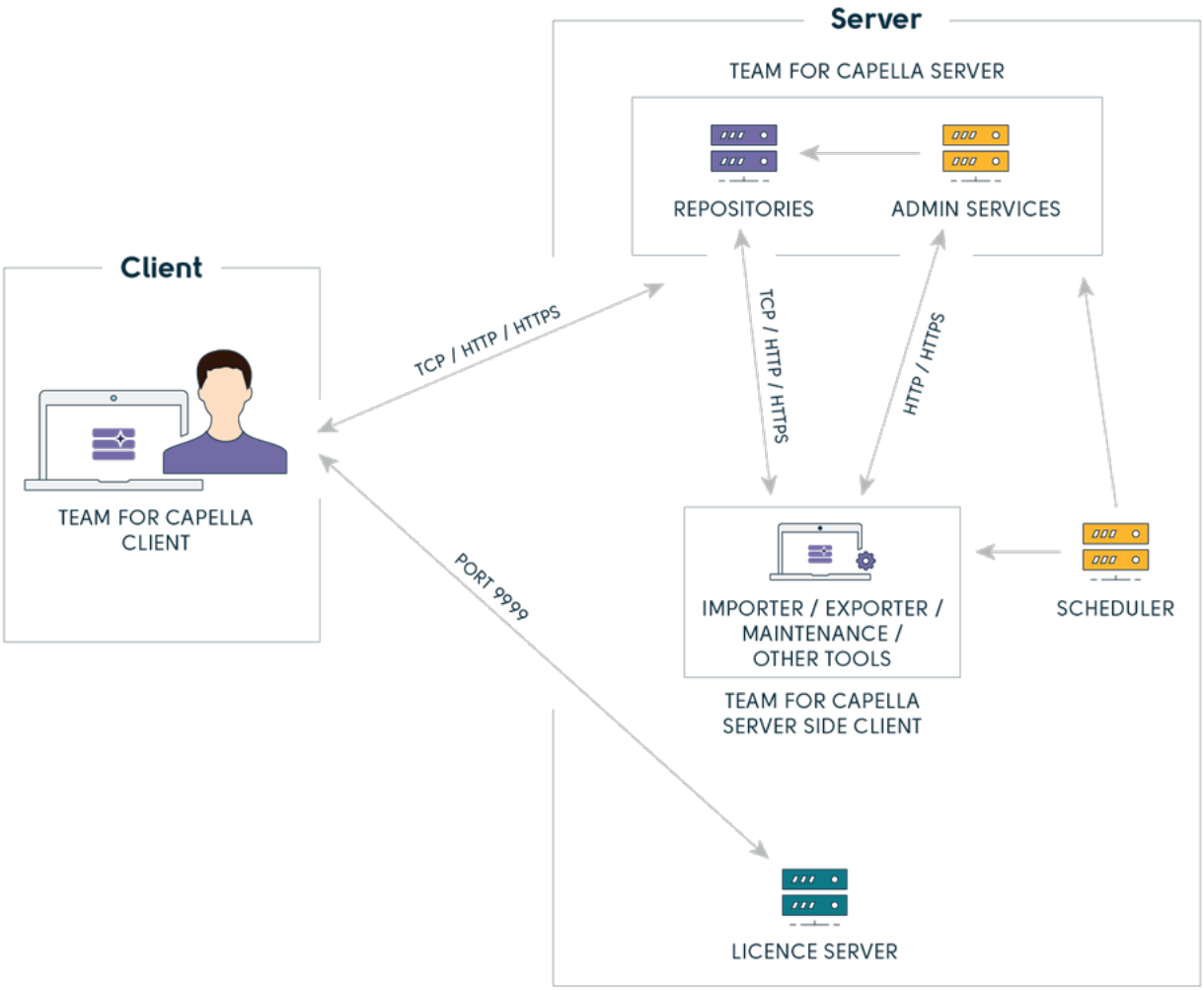
The truststore is protected with *secret* as a passphrase.

Team for Capella Server: the REST Admin Server

The Team For Capella server is composed of the CDO repositories server and an HTTP Jetty server.

By default, the Jetty admin server is automatically started with the CDO server on port **8080**. The admin server is used :

- to manage repositories with the REST Admin API
- by applications (importer, maintenance application, console application) to execute code on server



You can find more information in the file **<TeamForCapella installation folder>/server/configuration/fr.obeo.dsl.viewpoint.collab.server.admin/admin-server.properties** : it contains all the admin server configuration information.

REST Admin API

The REST Admin Server provides a whole set of services to manage the projects, the models and the users. The documentation is available at the URL **http(s)://<admin server IP>:<admin server port>/doc**

A **swagger documentation** is available at the URL **http(s)://<admin server IP>:<admin server port>/openapi**. It can be enabled or disabled with the `admin.server.jetty.servlets.admin.docandopenapi.enabled` property

Credentials Management

The first time the server is launched, a default **«admin» user and its associated default token are created in the Eclipse secure-storage of the user that started the CDO server**. The «admin» credentials are stored in a dedicated node used by the server. The token is hashed and encrypted. A ***secret.txt* file**, containing the token, is created in the same folder than *admin-server.properties* file. It can be used in third party application to authenticated with the admin server. **Do not forget to remove this file** as soon as you can.

Moreover, the admin credentials are also added in the secure storage for the application needs (importer, exporter, etc) in a dedicated node. The credentials are encrypted.

This way **once the server has been started the first time, there is no additional step. The applications can automatically be used**, being authenticated with the admin server with the «admin» user.

Nevertheless, it is possible to manage the user and the user token with the **Credentials application**

By default, the secure storage is created or retrieved from the home of the system user currently executing the application:

- Windows:
  - `%USERPROFILE%\.eclipse\org.eclipse.equinox.security\secure_storage`
  - `C:\Users\someUser\.eclipse\org.eclipse.equinox.security\secure_storage`
  - Specific case when Jenkins service is launched as Local System (not recommended): `C:\Windows\System32\config\systemprofile\.eclipse\org.eclipse.equinox.security\secure_storage`
- Linux:
  - `~/.eclipse/org.eclipse.equinox.security/secure_storage`
  - `/home/someUser/.eclipse/org.eclipse.equinox.security/secure_storage`
- macOS:
  - `~/.eclipse/org.eclipse.equinox.security/secure_storage`
  - `/Users/someUser/.eclipse/org.eclipse.equinox.security/secure_storage`
  - Location can depend from the configuration of your operating system.

It is also possible to change the location of the secure storage with the use of the `-eclipse.keyring`

program argument in both `TeamForCapella/server/server.ini` and `TeamForCapella/capella/capella.ini`. The secure storage must be shared between server-side client, tools and server in order to be able to use it from the Scheduler jobs. For example to use a fixed secure storage located in `TeamforCapella/.eclipse/secure_storage`:

```
-eclipse.keyring
../.eclipse/secure_storage
```

The `-eclipse.password` option can be used with `-eclipse.keyring`, it allows to use the content of the specified file as the default password instead of saving the password in the secure storage. On Linux, this helps to overcomes issues related to passwords management by secure storage and the use of the default PasswordProvider.

```
-eclipse.password
../.eclipse/password
```

Server logging

The server produces several log files by default:

- standard output is logged in a file created at each server launch **<TeamForCapella installation folder>/server/configuration/<timestamp>.log**
- exploitation logs are logged in a file created **<TeamForCapella installation folder>/server/logs/server.log**. A rolling strategy is in place and file put archived and compressed logs in the **archived** subfolder.

The standard output log can be modified via several properties:

- `osgi.logfile`: if not set, default log file will be created in **<TeamForCapella installation folder>/server/configuration/<timestamp>.log**. If the value is a relative path, the parent path will be the configuration folder.
- `eclipse.consoleLog`: if "true", any log output is also sent to Java's System.out (typically back to the command shell if any). Handy when combined with -debug. (`-consoleLog` on the command line is equivalent to `eclipse.consoleLog=true` in **config.ini** or `-Declipse.consoleLog` in `-vmargs` section or **server.ini**.
- `eclipse.log.enabled`: "true" by default, setting it to "false" will disable eclipse built-in log writer.
- `eclipse.log.level`: sets the level used when logging messages to the eclipse log.
  - `ERROR` - enables logging only `ERROR` messages.
  - `WARNING` - enables logging of `ERROR` and `WARNING `messages.
  - `INFO` - enables logging of `ERROR`, `WARNING` and `INFO` messages.
  - `ALL` - enables logging of all messages (default value)
- `eclipse.log.backup.max`: the max number of backup log files to allow. The oldest backup log file will be deleted after the max number of backup log files is reached as a result of rotating the log

file. The default value is "10". A negative or zero value will cause the default value to be used.

- `eclipse.log.size.max`: the max size in Kb that the log file is allowed to grow. The log file is rotated when the file size exceeds the max size. The default value is "1000". A negative value will cause the default value to be used. A zero value indicates no max log size.
- `osgi.framework.useSystemProperties`: can be put in `-vmargs` section to indicate to look for previous properties in the `-vmargs` section and not in `config.ini` file.
- usage:
  - use -DpropName=propValue as a VM argument to the Java VM
  - set the desired property in the config.ini file in the appropriate configuration area

Regarding the exploitation log, the server can log actions through its parent logger "fr.obeo.dsl.viewpoint.collab.server" or using a collection of specialized loggers (listed below) to limit by nature the processes that should be logged.

This logging feature is activated and configured in a `logback.xml` file that is referenced by the server argument:

```
-Dlogback.configurationFile="path_to_logback.xml"
```

In this configuration file can be declared loggers that log different kind of information. Here is a list of log IDs that are in use:

- `fr.obeo.dsl.viewpoint.collab.server.operation.app`: operations on the server application (start, stop...)
- `fr.obeo.dsl.viewpoint.collab.server.operation.rest.admin.repo`: operations on repositories (list, creation, deletion...)
- `fr.obeo.dsl.viewpoint.collab.server.operation.rest.admin.user`: operations on users (list, creation, deletion...)
- `fr.obeo.dsl.viewpoint.collab.server.operation.rest.admin.others`: miscellaneous operations on the server (tasks, monitoring...)
- `fr.obeo.dsl.viewpoint.collab.server.security.net4j`: authentications with the server (FileManager, LDAP, OIDC, UserProfile)
- `fr.obeo.dsl.viewpoint.collab.server.security.rest.admin` : authentications with the REST API
- `fr.obeo.dsl.viewpoint.collab.server.net4j.session`: operations on shared session (opening, closing...)
- `fr.obeo.dsl.viewpoint.collab.server.net4j.commit`: commit operations on shared session
- `org.eclipse.jetty.server.RequestLog`: requests received on the Jetty Web Server (set to `off` by default)

The server will log at the `ERROR` level for errors, `INFO` level for operation start and `TRACE` level for operation success.

The proposed configuration, pattern, and appenders can be modified in the `logback.xml` file.

By default, those logs are appended to the console output and added to **<TeamForCapella installation folder>/server/logs/server.log**.

Team for Capella Server Installation Types

Quick Installation (1 Server, 1 Repository)

Installation process and details are described in the ***Installation Guide*** for Team for Capella.

Ask to viewpoint providers whether their viewpoint is compatible with Team for Capella. If the viewpoint is compatible with Team for Capella, deploy the viewpoint on every Team for Capella client and the importer used by server. Clean and export models again after a viewpoint installation.

- Drawbacks:
  - When the only repository is stopped, no project is available. This can happen when one model is detected as corrupted by the "Projects – Import – repoCapella" job which has the `-stopRepositoryOnFailure true` parameter in its default command.
  - User accounts are linked to a repository instance (if an user has an account on the repository, he can connect to all models of the single repository).

Configuration with 1 Server, n Repositories, N Models

Introduction

This is the recommended configuration to work with several projects.

- Advantages:
  - Only one instance of the Team for Capella Server (RAM consumption is limited),
  - Configuration/Management is easier than with n servers,
  - This configuration looks like the one used for DOORS and Git,
  - Repositories are independent: a repository can be removed without impact on the other repository(ies) (requires to stop the server for static repositories, can be done from the REST API for dynamic repositories),
- Drawbacks:
  - Admin / REST API accounts are linked to a server instance, an administrator can perform operation on all repositories of the server instance.
  - When the only server is stopped, no project is available, for example, during a maintenance operation. This can also happen when one model is detected as corrupted by the "Projects – Import – repoX" job if the `-closeServerOnFailure true` parameter is used,
  - Some configuration must be done (unlike the default configuration).

How to Add a New Repository

This section presents how to add a static repository: it is defined in `cdo-server.xml` and requires to restart the Team for Capella server to be taken into account.

> There is a second of repositories: dynamic repositories. They can be created with the use of the REST Admin Server and its REST API.

Hypothesis: the repository is added to a just installed version.

Add a new repository to the Team for Capella Server:

- Open **TeamForCapella\server\configuration\cdo-server.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<cdoServer>
    <!--acceptor type="http"/ -->
    <acceptor type="tcp" listenAddr="0.0.0.0" port="2036">
        <!--  <negotiator type="challenge" description="d:/hsqldb/users.properties"/>-->
    </acceptor>

    <repository name="repoCapella">

    <repository name="repoCapellaMem">

</cdoServer>
```

Note the 2 default repositories (content is collapsed in this screenshot),

- "repoCapella", this repository is stored in a data base(h2),
- "repoCapellaMem", this repository is stored in memory (it is only an example, this repository should not be used and can be deleted).

```
<repository name="repoCapella_newProject">
    <property name="overrideUUID" value="repoCapella_newProject"/>
    <property name="supportingAudits" value="false"/>
    <property name="supportingBranches" value="false"/>
    <property name="verifyingRevisions" value="false"/>
    <property name="currentLRUCapacity" value="10000000"/>
    <property name="revisedLRUCapacity" value="10000000"/>
    <property name="supportingEcore" value="true"/>
    <userManager type="auth"/>
    <store type="db">
        <mappingStrategy type="horizontal">
            <property name="toManyReferences" value="ONE_TABLE_PER_REFERENCE"/>
            <property name="toOneReferences" value="LIKE_ATTRIBUTES"/>
            <property name="qualifiedNames" value="true"/>
            <property name="forceZeroBasedIndex" value="true"/>
        </mappingStrategy>

        <jdbcDelegate type="preparedStatement">
            <!--
                to explicitly force prepared statement caching (e.g., if statement
                pooling is not supported by the JDBC driver, use <property
                name="cacheStatements" value="enabled" /> Else, the implicit
                default is: <property name="cacheStatements" value="guess" /> Which
                guesses the correct setting based on the JDBC driver's metadata.
                Also supported is the third setting "disabled".
            -->
        </jdbcDelegate>

        <dbAdapter name="h2-capella"/>
        <!--
            to use CLOB instead of VARCHAR to store description and documentation field : use the dbAdapter h2-capella instead of h2
            dbAdapter name="h2-capella"
        -->
        <dataSource class="org.h2.jdbcx.JdbcDataSource" uRL="jdbc:h2:db/h2/repoCapella_newProject/capella;LOG=1;CACHE_SIZE=65536;LOCK_MODE=0;UNDO_LOG=0" user="sa"/>
    </store>
</repository>
```

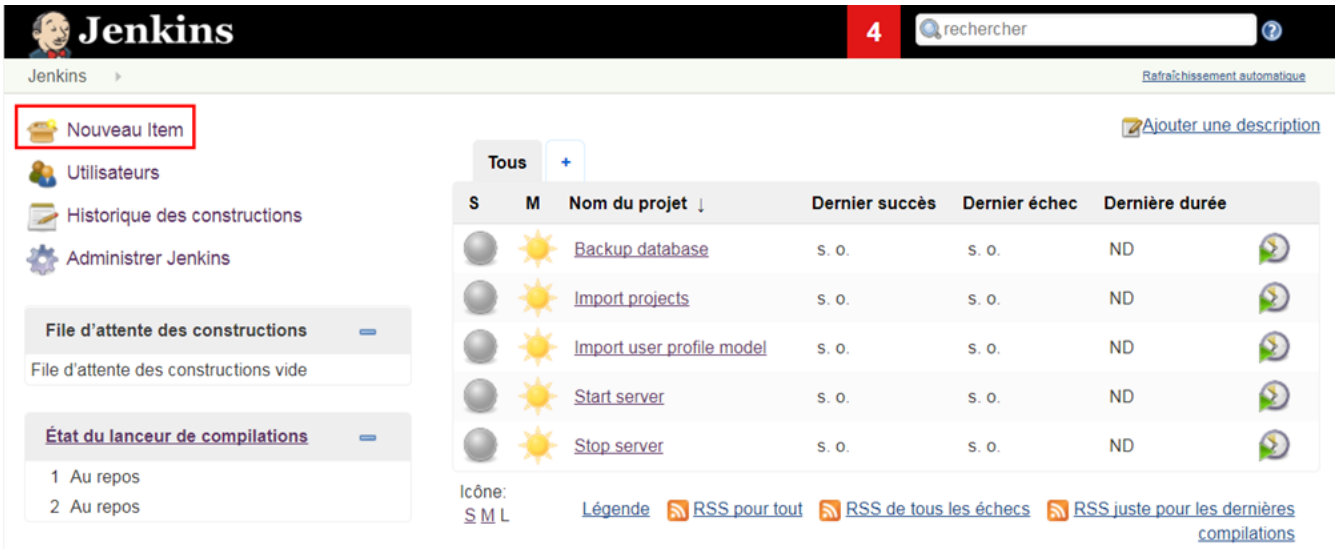
- The last changed argument gives the data base files names and location (here, files with be prefixed with "capella" and stored in "repoCapella_newProject" folder,
- For file backup/copy purposes, It is better to have each repository stored in its own folder,
- The default repository ("repoCapella") name and location should also be changed,

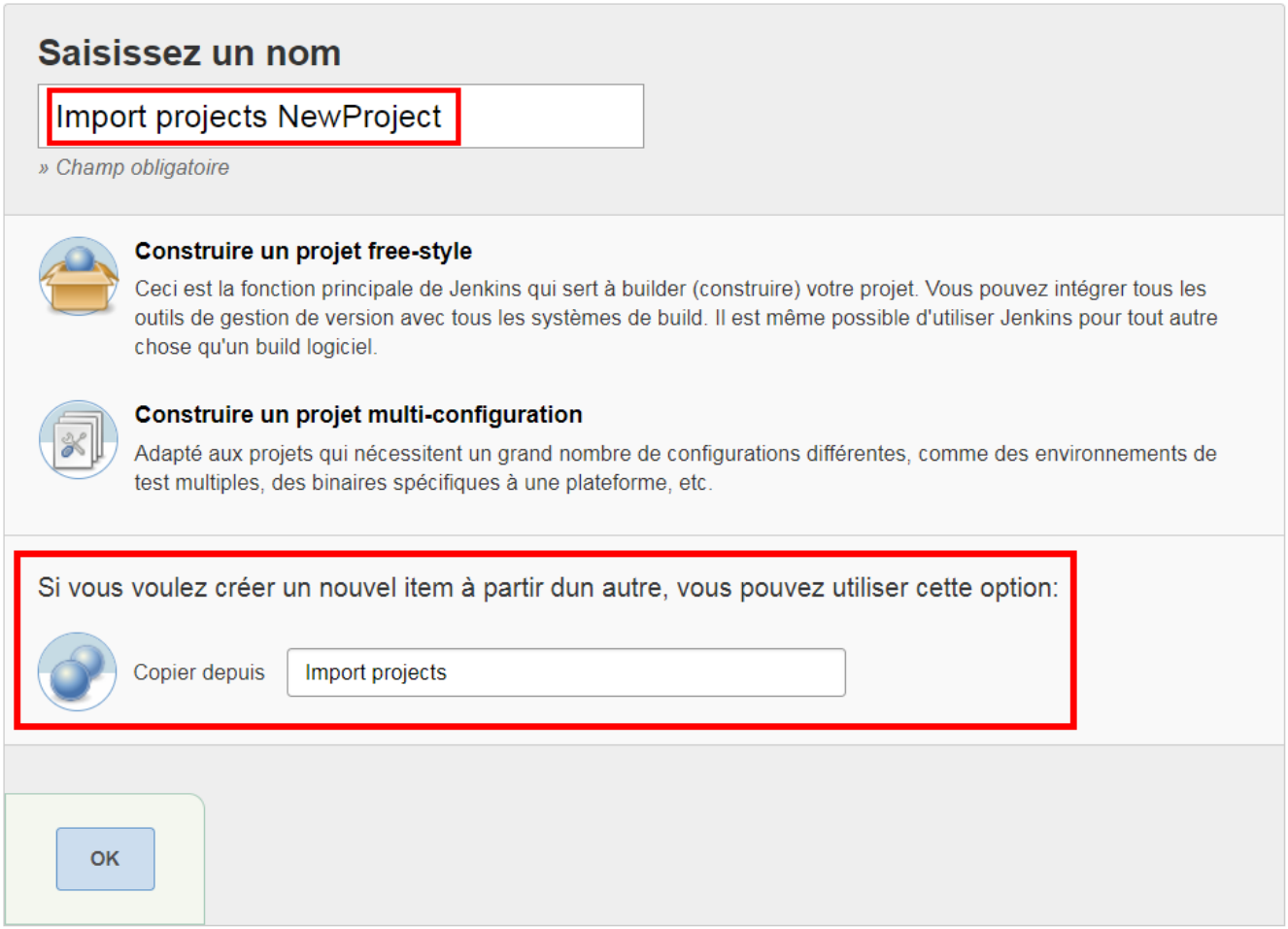
- Avoid space character in repository name (and generally special characters),

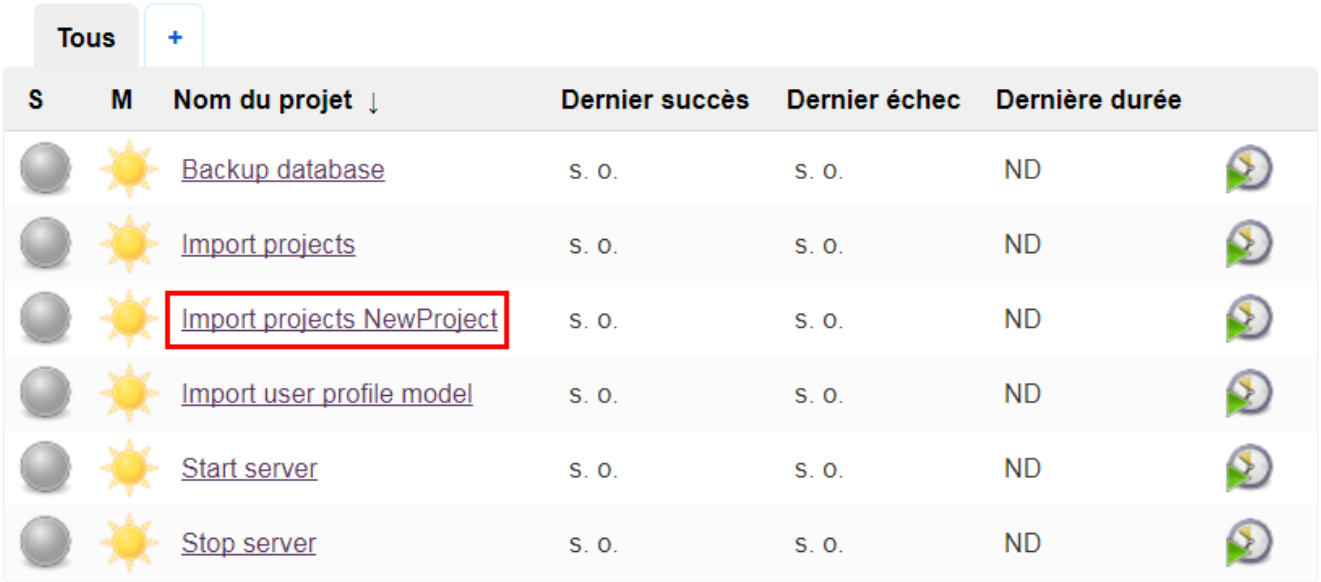
```
<?xml version="1.0" encoding="UTF-8"?>
<cdoServer>
    <!--acceptor type="http"/ -->
    <acceptor type="tcp" listenAddr="0.0.0.0" port="2036">
        <!--  <negotiator type="challenge" description="d:/hsqldb/users.properties"/>-->
    </acceptor>

    <repository name="repoCapella">
    ...
    <repository name="repoCapella_newProject">
    ...
    <repository name="repoCapellaMem">
    ...
</cdoServer>
```

Add a new job to Team for Capella Scheduler (Jenkins) to manage the new repository:

- Open the Scheduler (available by default at http://localhost:8036 , see Jenkins Configuration section for more details)

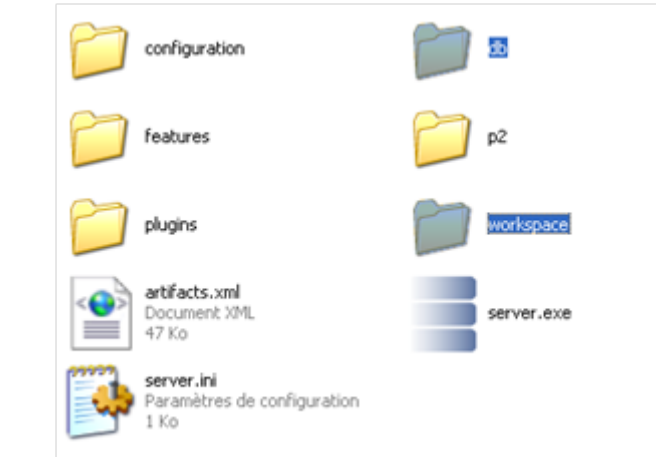


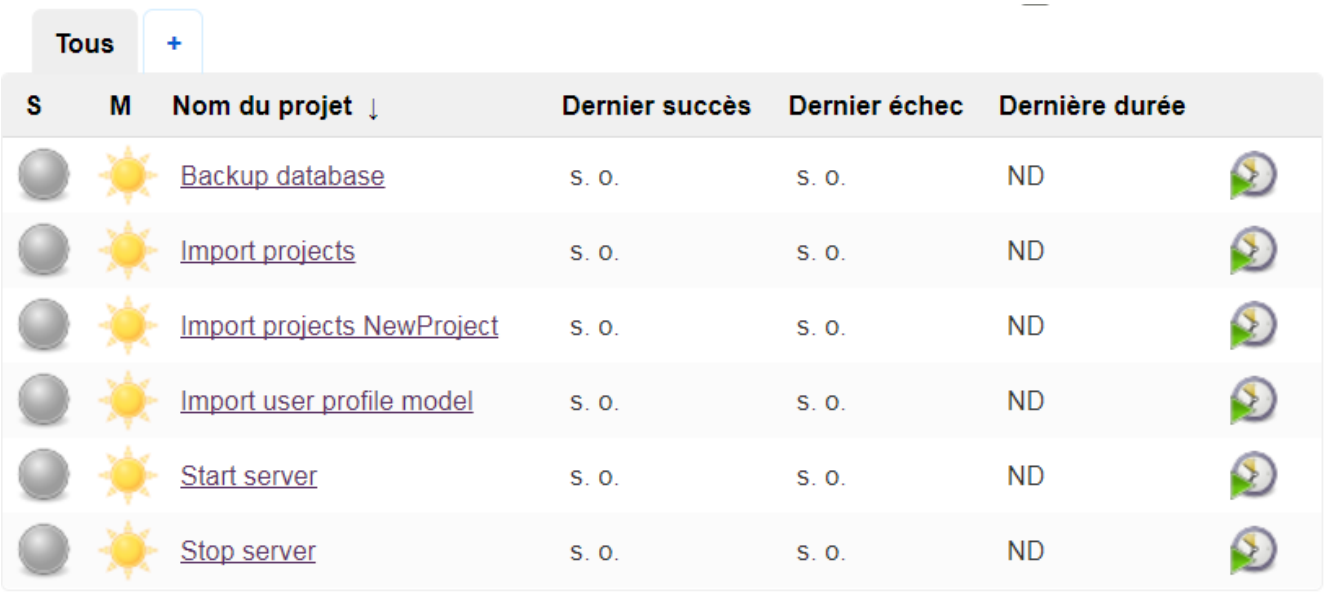




Check the configuration is working: Start the Team for Capella Server using the "Server – Start"job (click on )and open the **TeamForCapella\server** folder

db and **workspace** folders should have been created:





Configuration with N Servers, N Repositories, N Models (1 Scheduler)

Introduction

- Advantages:
  - If a server is stopped (for example if a database corruption occurs, and `-closeServerOnFailure true`), only projects stored on this server are unavailable,
  - Admin / REST API accounts can be different between server instances,
- Drawbacks:
  - Several instances of the Team for Capella Server (RAM consumption depends on the number on the number of servers used),
  - Configuration/Management is more complex than with only one server: duplication of ports, certificates,

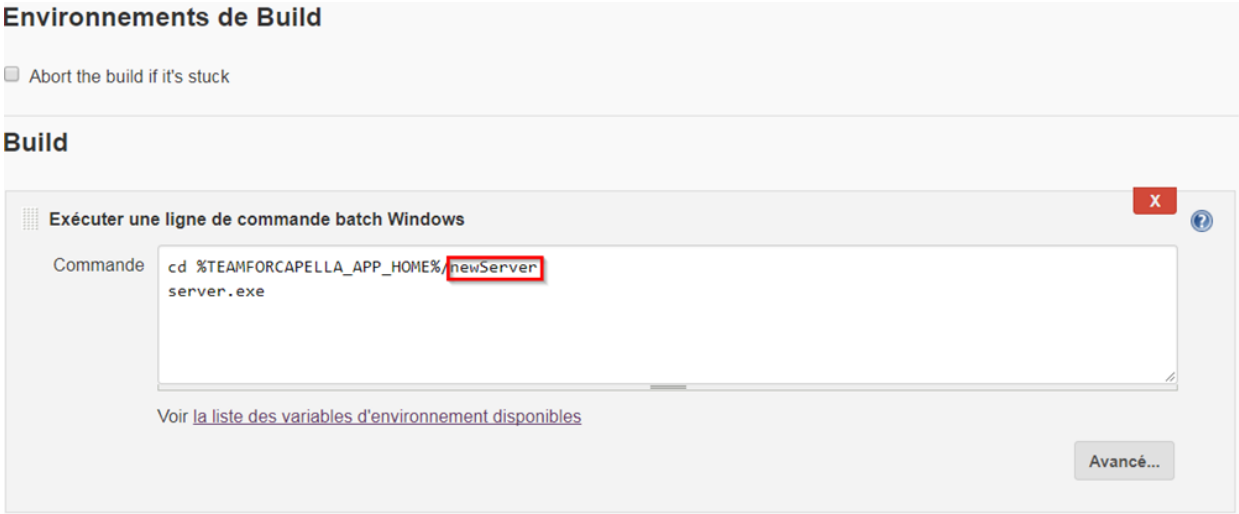
How to Add a New Server

Hypothesis: the server is added to a just installed version, by default it will only contain the default repository "repoCapella".

1. Create a new Team for Capella server instance,
   a. Do a copy of the **TeamForCapella\server** folder to **newServer** (for example),
   b. Change the cdo server port in the **TeamForCapella\newServer\configuration\cdo-server.xml**(for example 2037):

```
<cdoServer>
    <!--acceptor type="http"/ -->
    <acceptor type="tcp" listenAddr="0.0.0.0" port="2037">
        <!--  <negotiator type="challenge" description="d:/hsqldb/users.properties"/>-->
    </acceptor>
```

   c. Change the http server port in the **TeamForCapella\newServer\configuration\admin-server.properties**(for example admin.server.jetty.port=8081):
2. Add new jobs to Team for Capella Scheduler (Jenkins),
   a. Launch Jenkins,
   b. Using a web browser, connect to "http://localhost:8036",
   c. Duplicate all the jobs you need. (In Jenkins use «New item» button and fill «Copy from» field.)
   d. For every job, in the build part of the job, **add *-httpPort <admin server port>* parameter** to refer to the right instance of the admin server. (for example -httpPort 8081)
   e. For «Server - Run» job, in the build part of the job, change the path of the server



   f. «Backup and restore» and «Diagnostic and repair» jobs, in the build part of the job, **add *-port <cdo repository port>* parameter** to refer to the right instance of the cdo server(for example -port 2037)

How to stop the server

The main methods to close the server are the following:

- Launch the dedicated Scheduler job: Server – Stop (recommended method)
- directly *command.bat -command cdo -commandParams stopserver*

To avoid database corruptions, the server must in no way be closed these ways:

- Using the “Abort” button on the Server – Start job of the Scheduler,
- Especially on Windows 2008 Server 64 bits platforms:
  - Closing the command prompt running the server (if any) by clicking on the Windows close button,
  - Leaving the server close when the user logs out or the computer stops (to avoid this problem, it is advised to launch the Scheduler as a service so the server is not closed on log out).

How to reset the server

To restart with a clean server or after a database corruption, it can be useful to reset the server:

- Stop the server using the Scheduler,
- Remove the folder workspace from the server folder,
- Remove the folder db-auditing from the server folder (value for the default repository, check the dataSource elements of your cdo-server.xml file).
- Start the server,
- Export the needed models from a Team for Capella Client (using the “Import Job” result artifacts for example).

It is also possible to restore the database from the result artifacts of the Database – Backup job, refer to the Capella client Help Contents in chapter Team for Capella Guide > System Administrator Guide > Server Configuration > Reinitialize database.

The REST API `Start repository` and the `Repository - Start` job have a parameter

> allowing to drop al content of a repository on start.

How to Improve Export Performances

The following line is used to configure the database (in cdo-server.xm):

To improve performances when exporting big models to the repository, change LOG=1 by LOG=0. When exports are done, return to the original value (LOG=1 is useful to avoid database corruptions when the server process is killed).

Reinitialize database

You have three ways to reinitialize data in a database.

- Use the Database – Restore job
- Restoring a database backup
- Exporting backed up projects to a given repository

Restore database from database backup

The use of the Database – Restore job should be preferred but it is still possible to manually do the same operation.

This operation should be used to restore a database from the file generated by the Database – Backup job (this file has a pattern like: `repoCapella.20151105.171109-sql.zip`).

The database will be restored in exactly the same state as it was when the backup was performed:

- Existing durable locks will also be restored,
- A corrupted database will be restored in the same corrupted state.

How to manually restore a DB backup

1. Edit "server.ini" file
2. Change the vmarg property **collab.db.restore** to true as follow: ***-Dcollab.db.restore=true***
3. Specify the backup file location with the ***-Dcollab.db.restoreFolder*** parameter (default value is **db.restore** in the server)
4. Put the .zip backup file in the specified directory.

Example with **db.restore**:

1. Stop the server using the Server – Stop job
2. Start the server using the Server – Start job
3. If everything went well, you will get a log like the following one in the server's console:

```
!ENTRY com.thalesgroup.mde.melody.collab.server.repository.h2 1 0 2020-04-22
18:39:32.409
```

```
!MESSAGE Restore repoCapella processing starts.

!ENTRY com.thalesgroup.mde.melody.collab.server.repository.h2 1 0 2020-04-22
18:39:33.977
!MESSAGE Restore repoCapella restored database from :
C:\TeamForCapella\server\..\scheduler\jenkins_home\jobs\Database -
Backup\builds\7\archive\repoCapella.20200422.182742-sql.zip

!ENTRY com.thalesgroup.mde.melody.collab.server.repository.h2 1 0 2020-04-22
18:39:33.980
!MESSAGE Restore repoCapella processing ends. The file has been moved to
C:\TeamForCapella\server\..\scheduler\jenkins_home\jobs\Database -
Backup\builds\7\archive\repoCapella.20200422.182742-sql.zip.restored

!ENTRY org.eclipse.emf.cdo.server.db 2 0 2020-04-22 18:39:35.537
!MESSAGE Detected crash of repository repoCapella

!ENTRY org.eclipse.emf.cdo.server.db 1 0 2020-04-22 18:39:35.614
!MESSAGE Repaired crash of repository repoCapella: lastObjectID=OID248,
nextLocalObjectID=OID9223372036854775807, lastBranchID=0,
lastCommitTime=1 586 948 133 861, lastNonLocalCommitTime=1 586 948 133 86
```

The `.zip` backup file will be suffixed by `.restored` or `.error` if the restore failed. This behavior can be disabled with the use of ***-Dcollab.db.restore.rename.source.file=false***.

Restore process only supports textual script backup with the name that ends with –sql.zip.

If you want to remove restored locking sessions from the database, use the Durable Locks Management view (see the Server Administration part of this documentation).

Restore database from projects backup

This way gives more control on the restoration as you may delete the repository and the repository is restored project by project. To restore projects in a repository:

- close the server
- delete file corresponding to the repository in the database folder. See how you configured the **cdo-server.xml** file to have the information.
- restart the server
- export the projects to the server with a Team For Capella client. Those projects are taken from the last valid «Projects - Import» job execution.

How to externalize configuration in a specific folder

- To externalize **workspace** → use the eclipse runtime arguments *"**-data** path_to_folder "* in the files **capella.ini**, **importer.bat** and **command.bat**.

Example:

```
server/server.exe -data C:/data/TeamForCapella/server/workspace
```

```
capella/importer.bat -data C:/data/TeamForCapella/server/importer-workspace
```

```
capella/command.bat -data C:/data/TeamForCapella/server/command-workspace
```

- To externalize configuration folder → copy the folder configuration to the expected path and use the eclipse runtime arguments *"**-configuration** path_to_folder "*.

Example:

```
server/server.exe -configuration C:/data/TeamForCapella/server/configuration
```

```
tools/importer.bat -configuration C:/data/TeamForCapella/server/configuration
```

```
tools/command.bat -configuration C:/data/TeamForCapella/server/configuration
```

- To externalize **cdo-server.xml** → use the jvm arguments from the **server.ini** *"**-Dnet4j.config**= path_to_file "* .

Example:

```
-vmargs -Dnet4j.config=C:/data/TeamForCapella/server/configuration/cdo-server.xml
```

- To externalize **users.properties** → update the **description** property of the <**usermanager**> element from the cdo-server.xml file.

Example:

```
Line 18 : <userManager type=«auth»
description="C:/data/TeamForCapella/server/usermanager-config.properties" />
```

- To externalize db folder → update jdbc url from the cdo-server.xml file editing the value of the attribute **url** from the tag <**dataSource**/>.

Example:

```
<dataSource uRL="jdbc:h2:C:/data/TeamForCapella/server/
```

```
db/h2/repoCapella;LOG=0;CACHE_SIZE=65536;LOCK_MODE=0;UNDO_LOG=0" (⋯)
```

- Move jenkins_home

Update scheduler/conf/context.xml to change the attribute Environment JENKINS_HOME with the path of the jenkins_home folder :

- Restart scheduler
- [Optional: in case you do not user embedded scheduler] To externalize backup and restore folder → use the jvm arguments from the **server.ini** file: *"**-Dcollab.db.backupFolder**= path_to_file "* and *"**-Dcollab.db.restoreFolder**= path_to_file "* .

Example:

```
-vmargs -Dcollab.db.backupFolder=C:/data/TeamForCapella/server/db.backup
```

```
-Dcollab.db.restoreFolder=C:/data/TeamForCapella/server/db.restore
```

To directly externalize all previous file, you can edit server.ini file

Example: To externalize all files in the folder **C:\data\TeamForCapella\server**

1) Update server.ini

```
-console
-data
 C:/data/TeamForCapella/server/workspace
-configuration
 C:/data/TeamForCapella/server/configuration
-vmargs
-Dnet4j.config= C:/data/TeamForCapella/server /configuration
-Dcollab.db.backup=false
-Dcollab.db.restore=false
-Dcollab.db.backupFolder= C:/data/TeamForCapella/server /db.backup
-Dcollab.db.restoreFolder= C:/data/TeamForCapella/server /db.restore
-Dcollab.db.backupFolderMaxSize=1G
-Dcollab.db.backupFrequencyInSeconds=900
-Dosgi.requiredJavaVersion=17
-Xms128m
-Xmx2000m
-XX:PermSize=128m
```

- Update "-data" and "-configuration" of command.bat and importer.bat

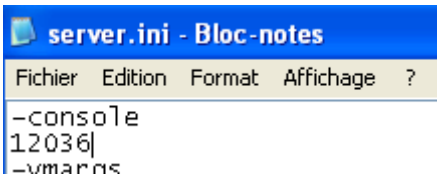
How to Change Ports Values

See Server configuration section → Cdo-server.xml File

See Jenkins installation section → Change the Port Used by Jenkins.

See Team For Capella Web server section → Change the Port of the admin server

By convention we could use 12036 for a server that listens to the port 2036 (defined in cdo-server.xml), 12037 for the server that listens to 2037, 12038 for 2038 etc...



- Edit configuration of all jobs setup in the scheduler that use the OSGI console
- Connect to scheduler admin site
- Go to "Backup database" configuration
- Edit the build command line to replace "...command.bat localhost port_value" by the expected port

Ex: command.bat localhost 12036 capella_db backup

- Go to "Server – Stop" configuration
- Edit the build command line to replace "...command.bat localhost 12036 close" by the expected port

Ex: command.bat localhost 12036 close

- Go to "Import projects" configuration
- Edit the build command line to replace "...importer.bat –archivefolder..." by the expected port

Ex: importer.bat –consoleport 12036 –archivefolder

- This is needed if the importer has to stop the server on import failure

If you have several jobs using the OSGI port value, you can create an environment variable to store it in a single place.

How to Increase the Size of Description and Documentation Columns

When very long text are written in Description or Documentation fields, an error of the following type can occur when saving a remote project or exporting a local project to the server:

```
[ERROR] org.h2.jdbc.JdbcSQLException: Value too long for column DESCRIPTION
VARCHAR(32672)
```

To avoid this problem, change the file server/configuration/cdo-server.xml to use:

<**dbAdapter name="h2-capella"** /> instead of <dbAdapter name="h2" />

Fields description and documentation will be stored in CLOB instead of VARCHAR.

h2-capella is the default value in cdo-server.xml.

Customize commit description length

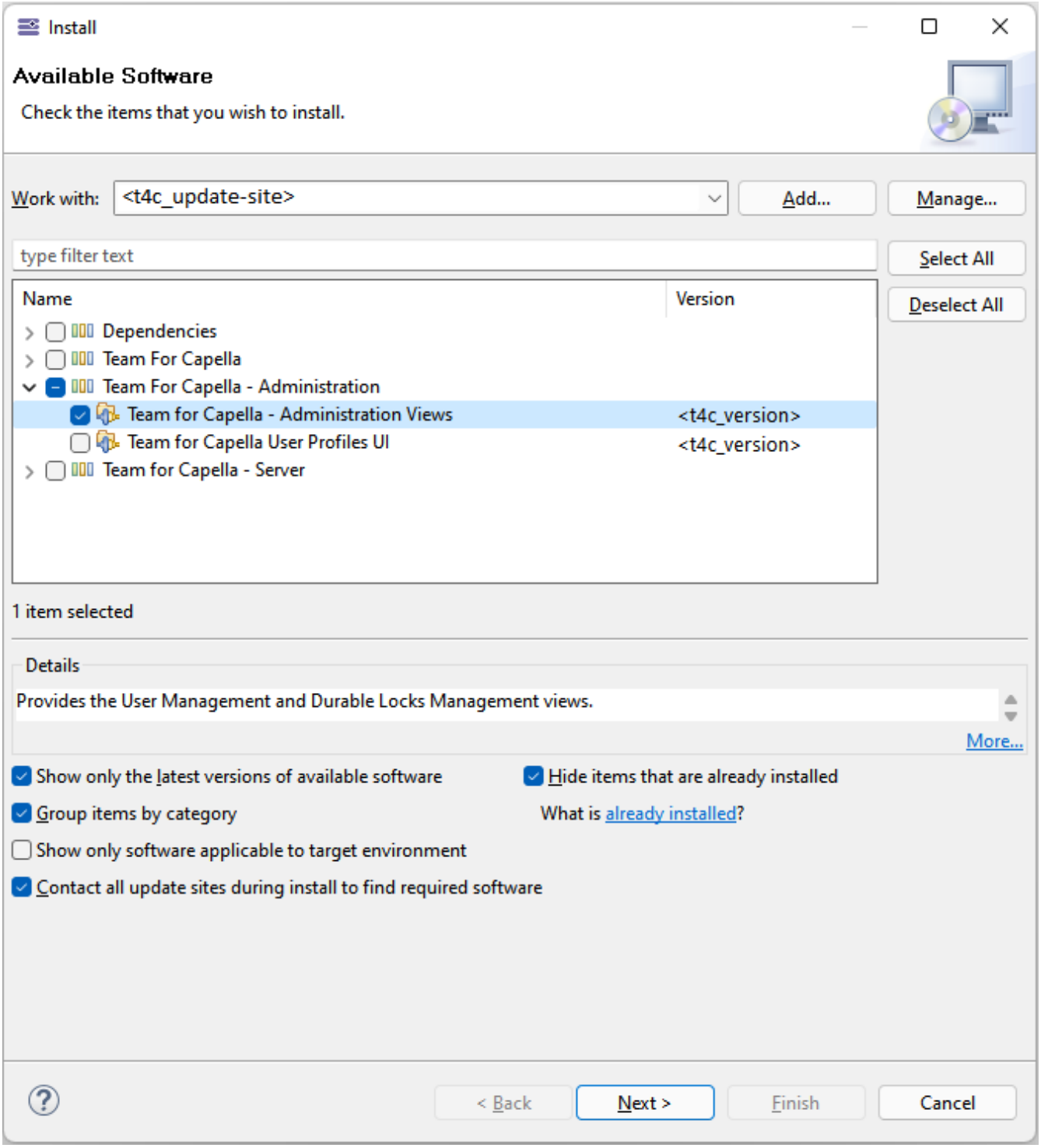
The system property `fr.obeo.dsl.viewpoint.collab.common.commit.description.max.length` can modify the length of the commit message. By default the value is the Integer maximum value. **This property needs to be set to the same value on the client and the server**.

This default Integer maximum value length is due to com.thalesgroup.mde.melody.db.h2.H2Adapter that consider the commit comment as a CLOB. Note that this is a custom H2Adapter for Team for Capella. It replaces the default org.eclipse.net4j.db.h2.H2Adapter that expects a VARCHAR for the comment description DB field limiting the length of the message to 255 characters. If the commit description is longer than the accepted max length, it will be truncated before commit to match the max length in core wizards, actions and session save operations. For components extending the collaborative layer, if they directly call setCommitComment() and commit() methods on the CDO transaction, they can use `fr.obeo.dsl.viewpoint.collab.common.internal.commit.CommitCommentUtil.fitCommitDescriptionLength(String)` to fit their commit comment. -1 can be used to remove the limitation. Otherwise only values greater than or equals to 10 are accepted. "Xxxxxxxxxx" will become "Xxxx [...]". If the property value if -1, it will take the system dependent SWT widget Text.LIMIT length.

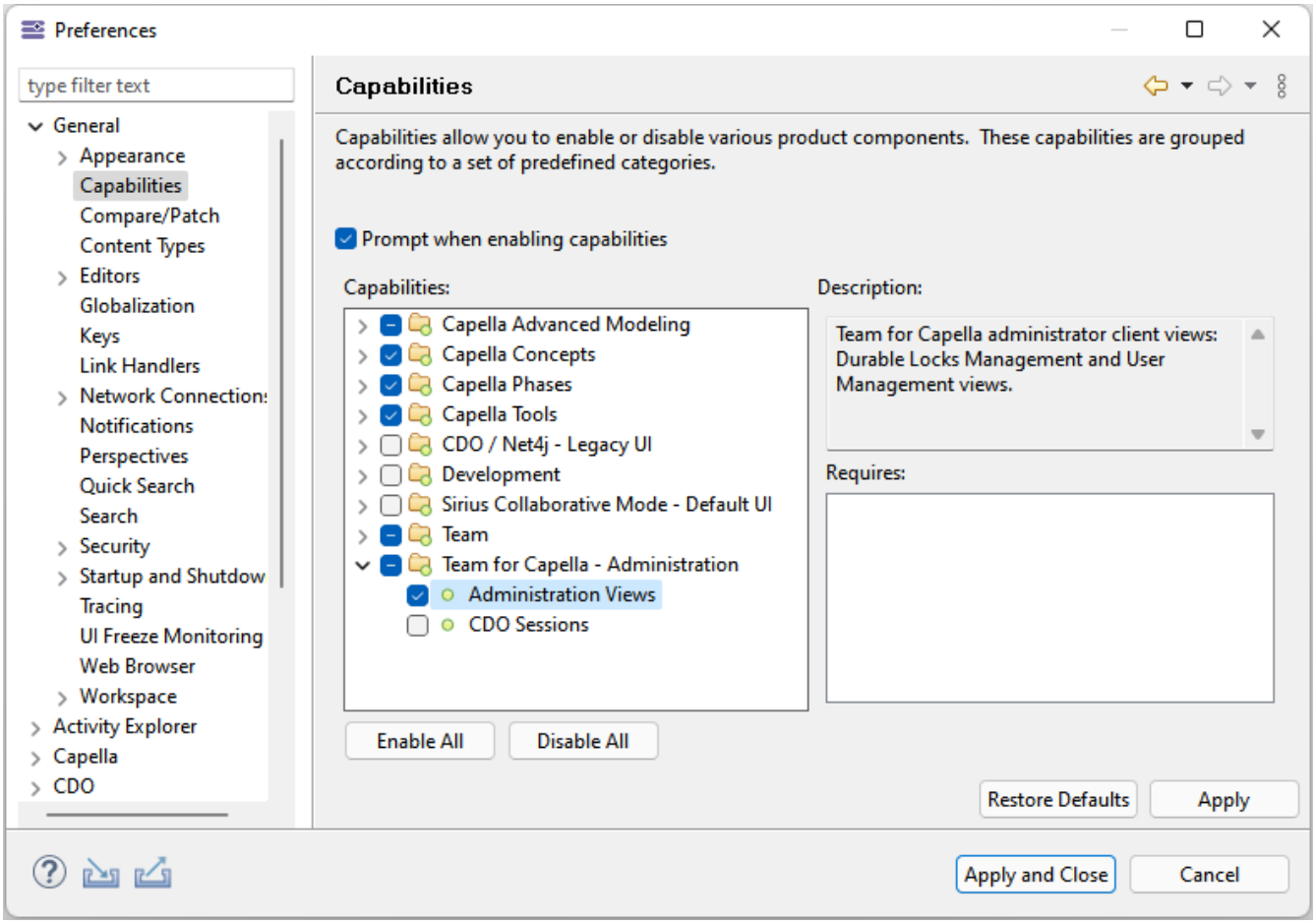
5.4. Server Administration

Administration Views

The Team for Capella client comes with two views useful to perform some administrative tasks: The **Durable Locks Management** view, and the **User Management** view. To access to these features, you must install the *Team for Capella - Administration Views* feature from the Team for Capella update site.



After restarting your T4C client, go to Preferences > General > Capabilities to enable the **Administration Views** capability.



Durable Locks Management View

Important: The durable locking is deactivated by default since Team For Capella 1.1.4 and 1.2.1.

Activate the durable locking

The durable locking mechanism allows to configure the explicit locks manually taken by a user as persistent locks. If a user takes explicit locks and then terminates his connection to the remote model (by closing his shared project or exiting the Team for Capella client), his explicit locks are not released and he will retrieve them on the next connection to the repository.

The durable locking can be activated by a client by adding the following option in the plugin_customization.ini file:

```
fr.obeo.dsl.viewpoint.collab/PREF_ENABLE_DURABLE_LOCKING=true
```

If the plugin_customization.ini file is not present, you need

- to create it in `capella/`
- to reference it from the `capella/capella.ini`: before `-vmargs`, add:

```
-pluginCustomization
```

```
plugin_customization.ini
```

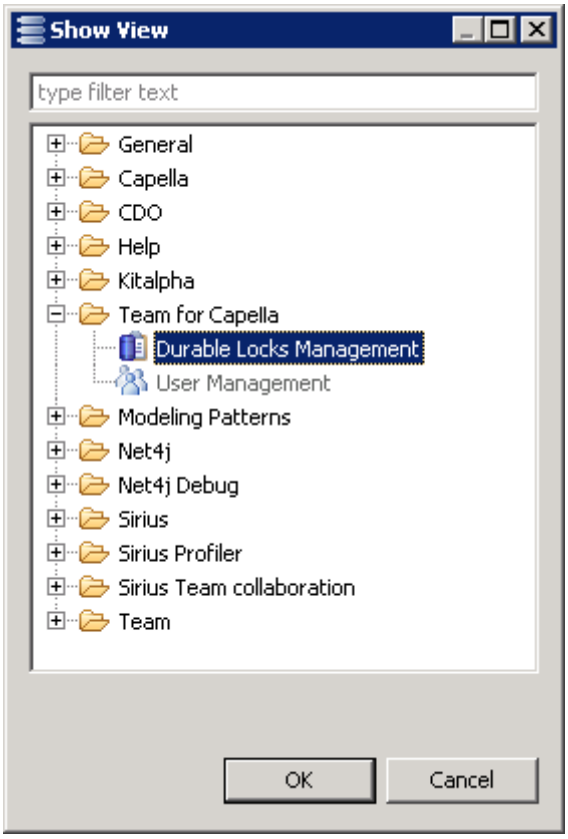
Note that the activation or deactivation of durable locking will have no effect on existing connection projects. The client have to remove the local connection project and to connect to the remote project again.

The following sections describe the case where the durable locking is activated.

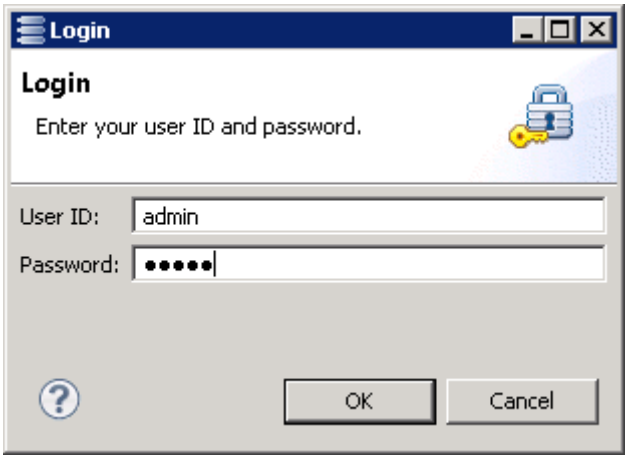
Use the View

Team for Capella provides the Durable Locks Management view to list existing locking sessions and delete them if needed.

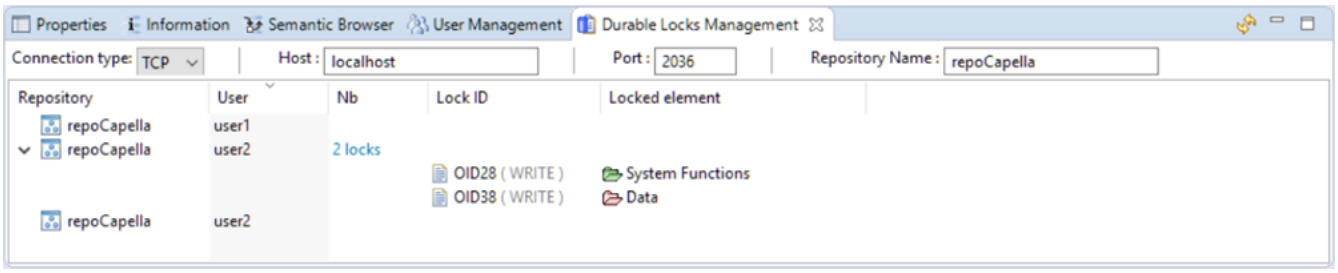
- To open this view in a Team for Capella client, click on Window / Show View / Other... and select **Team for Capella / Durable Locks Management**:



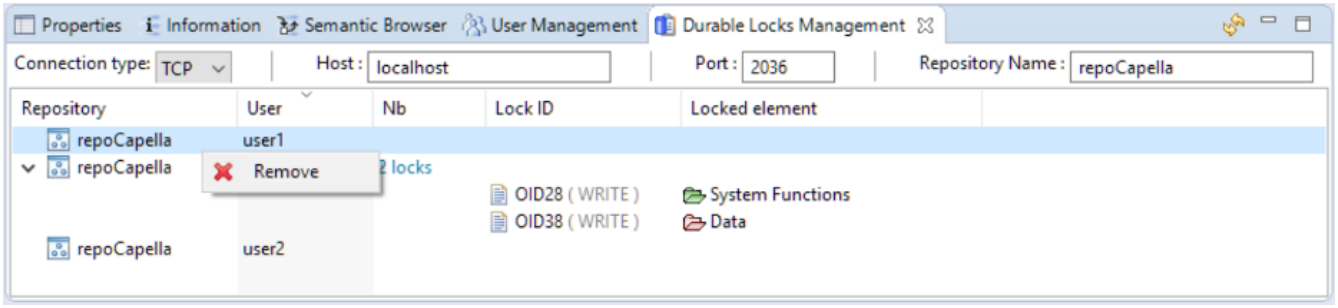
When doing the first operation with this view, you will be asked to logon with the following dialog:



- Click on the button to list existing locking sessions (object locked by these locking sessions, if any):



- Use the contextual menu on a Locking Session to remove it:



It is allowed to remove Locking Sessions only if the corresponding user is **not connected**.

Additional information on Locking Sessions

The Durable Locks Management view displays all locking sessions existing on the repository and the locks created by these locking sessions (if any).

A locking session is created whenever a team project is created on a client (Capella Connected Project). So if a user creates several team projects, he can have several locking sessions (as user1 in the screenshot above). Each locking session has a unique ID stored in the local .aird file.

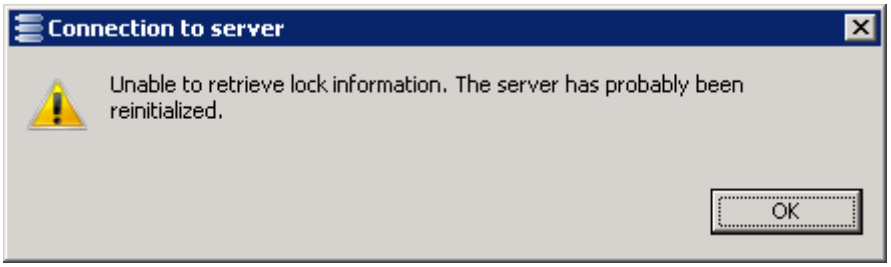
Locks are owned by a locking session, so if the same user has two locking sessions (2 team projects) and he locks an element in the first locking sessions, this element will appear with a red lock in the second locking session.

Remove Locking Sessions

As explained above, using the Durable Locks Management view, locking session can be removed (this action is available by all users but should be done by the administrator only). A locking session can be removed only if nobody is connected using it.

All locks hold by the locking session are removed with it.

If a user tries to connect to the repository using an existing connection project referencing a removed Locking Session ID, an error dialog is displayed (see below) and a new locking session is created. The ID of this new locking session will replace the old one in the local .aird file on the next save action.

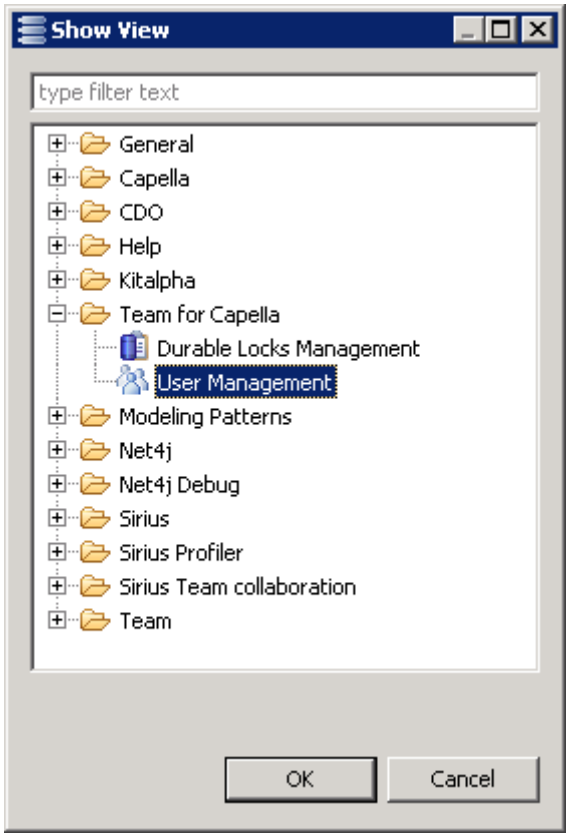


User Management View

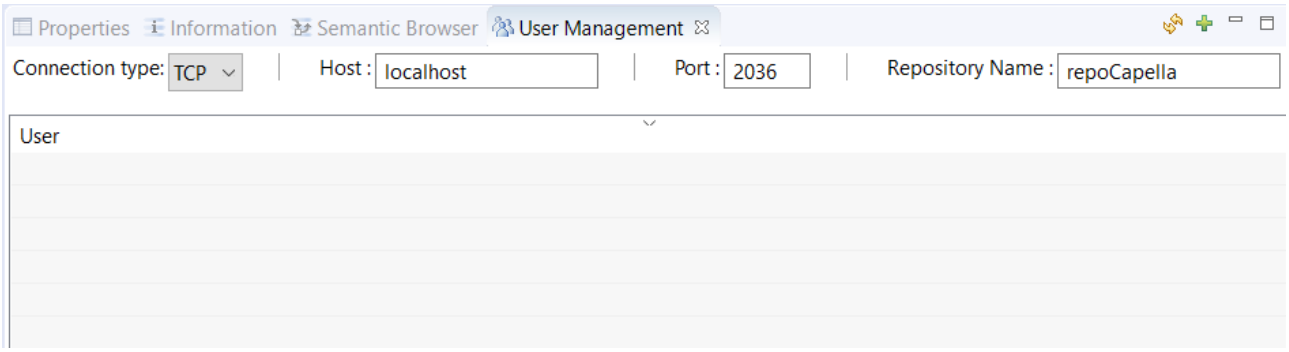
Team for Capella provides the User Management view to manage users on the Team for Capella Server.

The Durable Locks Management view is useful only if the Team for Capella Server is configured to work with the access control "**Identification**".

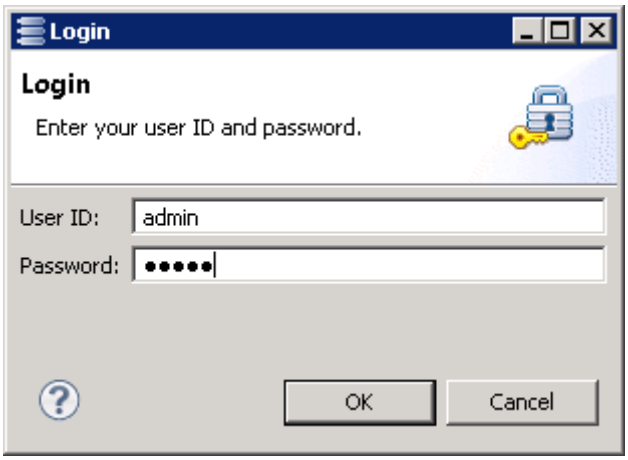
- To open this view in a Team for Capella client, click on Window / Show View / Other... and select **Team for Capella / User Management**.



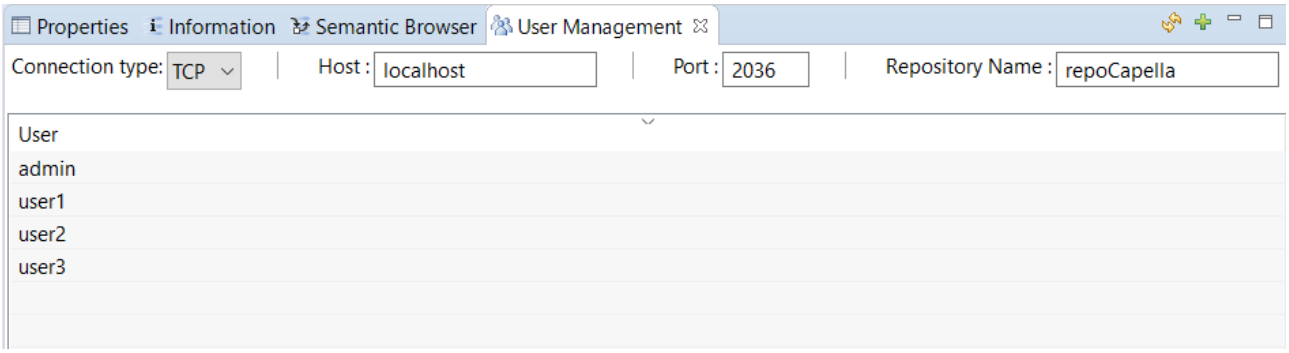
The view is shown.



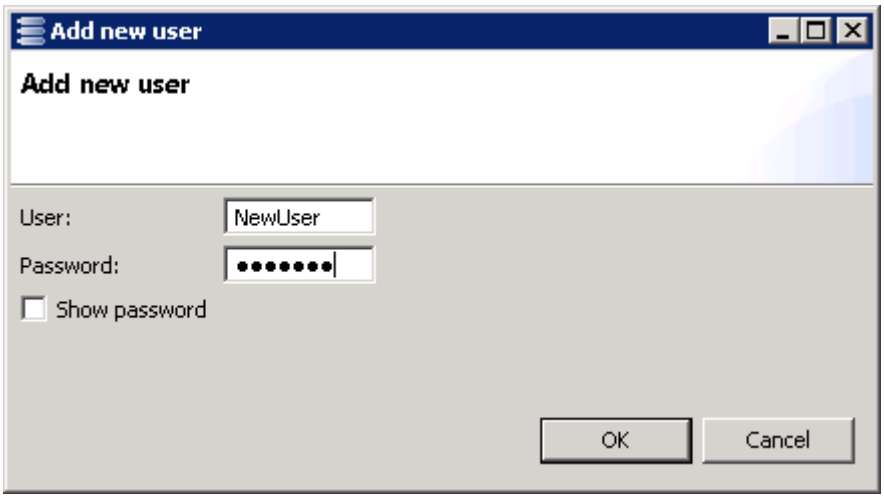
When doing the first operation with this view, you will be asked to logon with the following dialog:



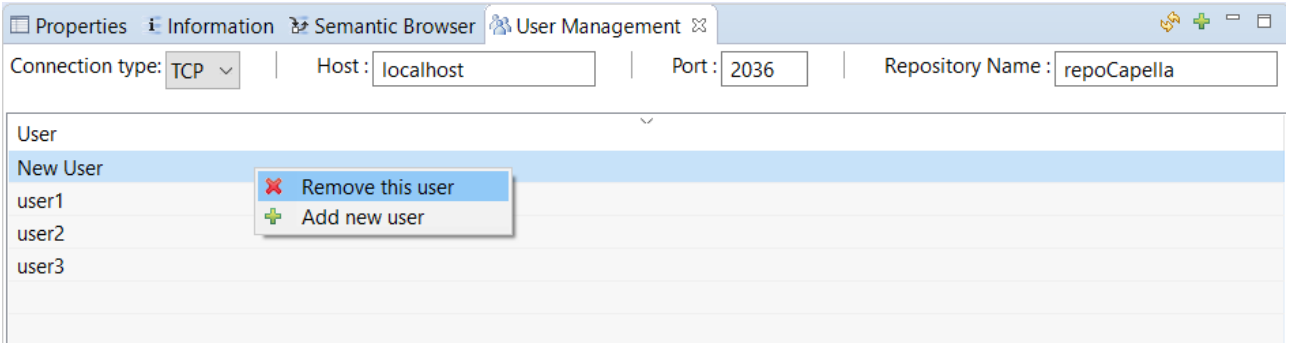
- Click on the  button to list already registered users:



- Click on the "+" button to add a new user:



- Use the contextual menu on a user to remove it:



Administration Tools

Repository maintenance application

The repository might have some inconsistent data and might need to be maintained.

The Repository maintenance application will look for the following inconsistencies:

- Broken links between Representation Descriptor and their representation(a Diagram, a table or a tree).

This link might be broken if the representation has been deleted or if the internal index of the Representation Descriptor list is incorrect. That can cause some troubles for the different users connected to the project.

- Stale references (Orphan references): Some references in the model might be linked to a missing element in the database. That might cause the importer failure. The diagnostic job will list them in the console log. If the repair is activated, stale references will be removed.

The application aims to delete orphan Representation Descriptors and stale references in the repository (both graphical and semantic models).

Once done the application will close the server.

This application requires that no user is connected to the repository.

Job configuration

There are two jobs available for maintenance in the Scheduler:

- *Start repository diagnostic* will only run the diagnostic part. The diagnostic result is logged in the console output of the job. It is also kept as an artifact of the job result.
- *Start repository maintenance* will run the diagnostic and then launch the maintenance tasks if some managed issues are detected: it will backup the server with capella_db command, perform the required changes on the database and close the server. The steps are logged in the console output of the job and the corresponding log file is kept as an artifact of the job result.

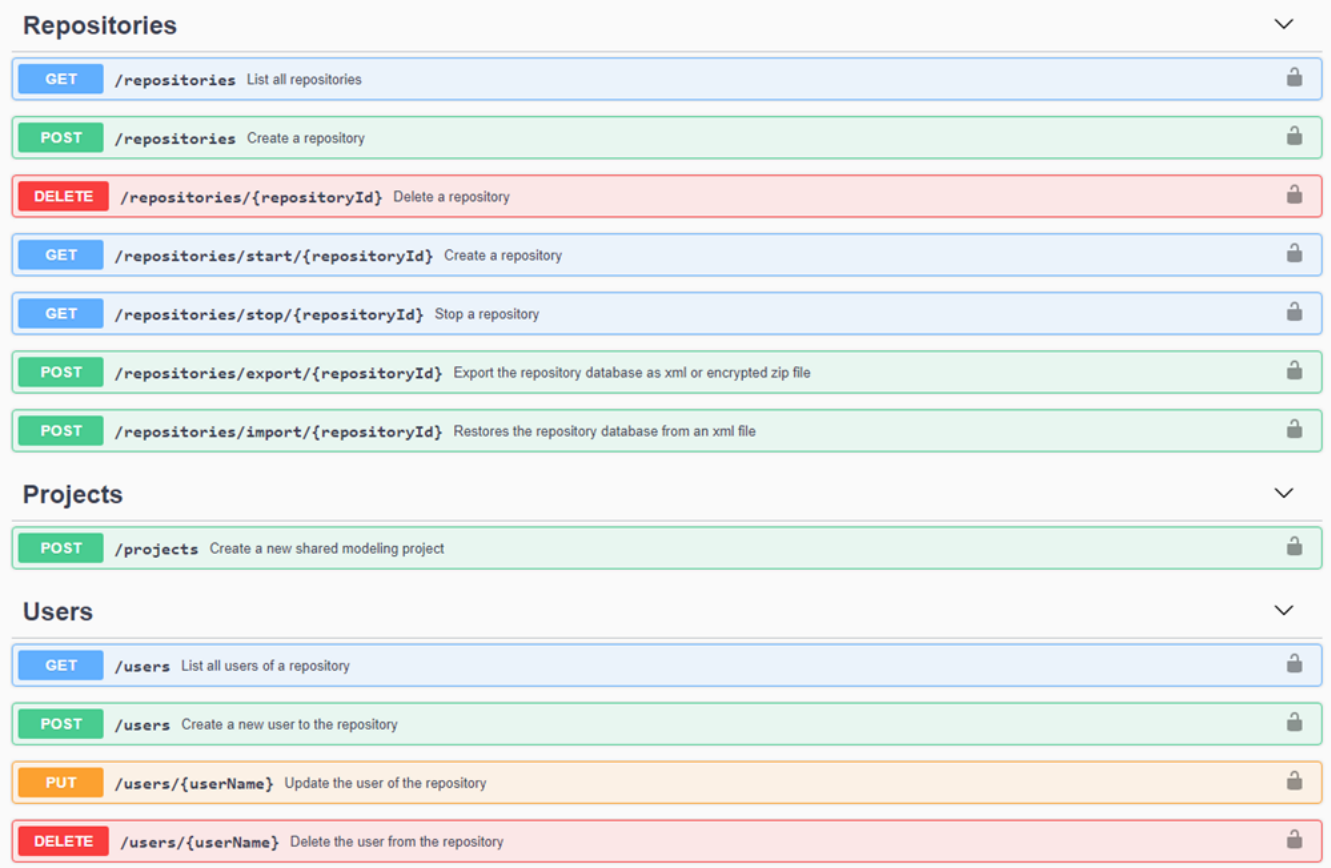
The application needs credentials to connect to the CDO server if the server has been started with authentication or user profile. Credentials can be provided using `-repositoryCredentials` parameter. Here is a list of arguments that can be passed to the application or using the job (in maintenance.bat or the job config):

Arguments	Description
-repositoryCredentials	Login and password can be provided using a credentials file. To use this property file • Add the following program argument: `-repositoryCredentials <path_to_credentials_file>` • Fill the specified file using the following format (only one line allowed): `aLogin:aPassword`
-hostname	defines the team server hostname (default: localhost).
-port	defines the team server port (default: 2036).
-repoName	defines the team server repository name (default: repoCapella).

Arguments	Description
-connectionType	The connection kind can be set to **tcp** or **ssl** (keep it in low case) (default: tcp)
-diagnosticOnly	Allowed values are **true** or **false**. If true, only the diagnostic is done. The database will be unchanged. (default: false)
-launchBackup	Allowed values are **true** or **false**. If true, the capella_db backup is done before any change is done on the database. (default: true)
-archiveFolder	Indicates where the backup zip will be stored.
-httpLogin	Backup and Maintenance are triggered by an Http request. This argument allows to give a login to identify with on the Jetty server.
-httpPassword	Backup and Maintenance are triggered by an Http request. This argument allows to give a password to authenticate with on the Jetty server.
-httpPort	Backup and Maintenance are triggered by an Http request. This argument allows to give a port to communicate with on the Jetty server.
-httpsConnection	Backup and Maintenance are triggered by an Http request. This boolean argument specifies if the connection should be Https or Http.

REST Admin Server

An administration feature through WebServices is available for the Team for Capella Server: it brings users and repositories management capabilities through REST API and exposes an OpenAPI description:



Refer to documentation available in the folder server/dynamic to discover how to install and enable it.

5.5. Access Control (User Profiles)

Available Access Control Modes

Several modes of access control can be used for each repository on the server:

- "Identification" (default mode):
  - Each user defined in the file user.properties is authorized to read and/or modify all models present on the repository.
  - Refer to *Server Configuration/Authenticated Configuration*
- "User Profiles":
  - Discriminating user rights are defined in a User Profiles model.
  - Refer to *Server Configuration/User Profiles Configuration*
- "LDAP Authentication":
  - This mode allows authenticating with an LDAP server. It can be also used with authenticated or with user profiles.
  - Refer to *Server Configuration/Activate LDAP Authentication*
  - Refer to *Server Configuration/Authenticated Configuration*
  - Refer to *Server Configuration/User Profiles Configuration*

- "OpenID Connection Authentication":
  - This mode allows authenticating using the UI provided by the OpenID Connect Platform. It can be also used with authenticated or with user profiles.
  - Refer to *Server Configuration/Activate OpenId Connect Authentication*
  - Refer to *Server Configuration/Authenticated Configuration*
  - Refer to *Server Configuration/User Profiles Configuration*
- "Not Authenticated Access":
  - Anyone can read and/or modify all models on the repository.
  - Refer to *Server Configuration/Not Authenticated Configuration*

Notices when configuring Access Control mode

Switching between different access control modes

When switching between different access control modes, the server must be restarted. Otherwise, the configuration update will not be taken into account.

User Profiles

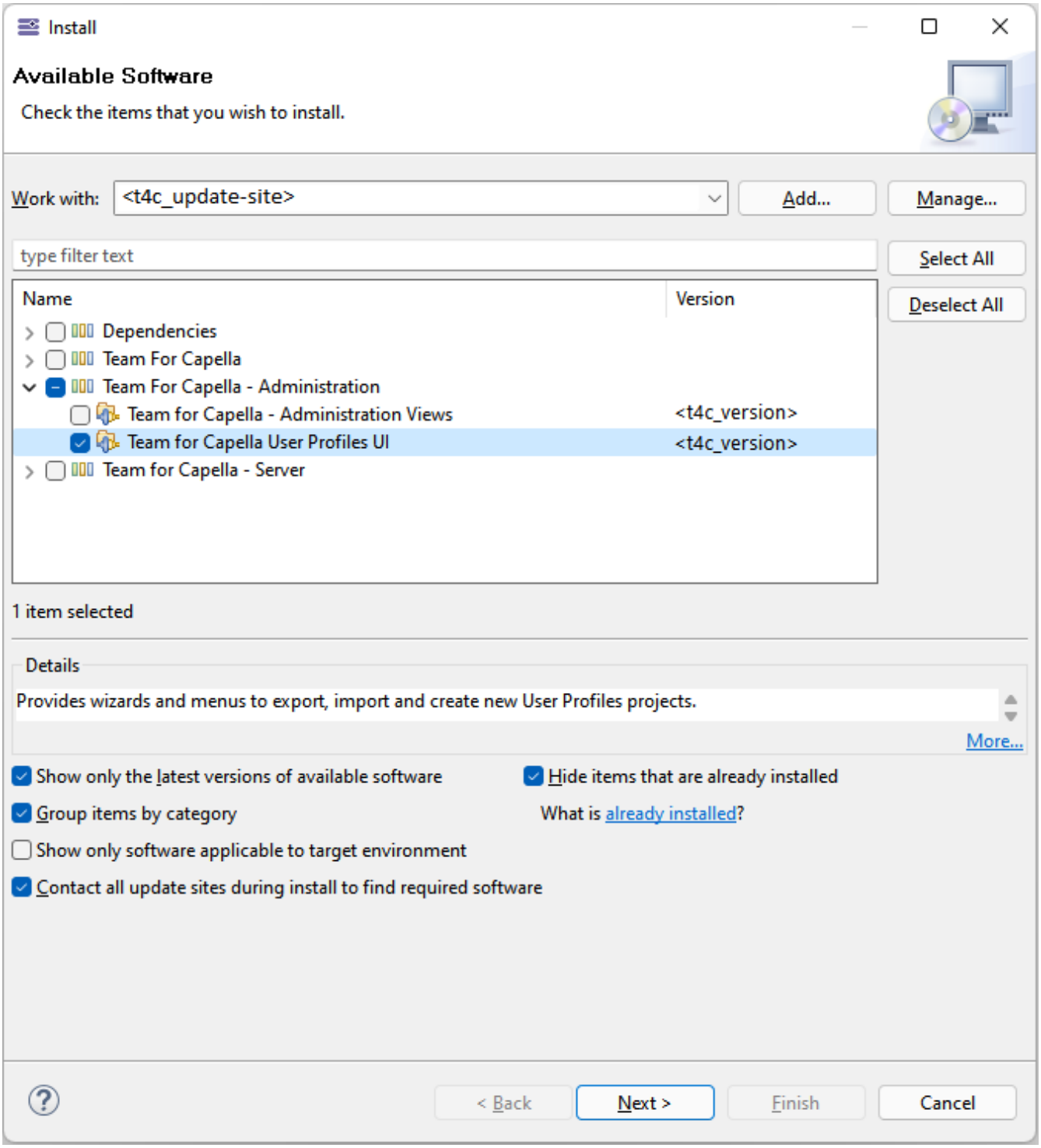
Configuration

In Team for Capella, when using the User Profiles feature, usernames and access rights are stored in the repository (i.e., in the database). Note that when passwords are stored in the user profiles model (when LDAP is not used), they are not encrypted. That's why the usernames management part of this feature must be considered as a simple identification feature.

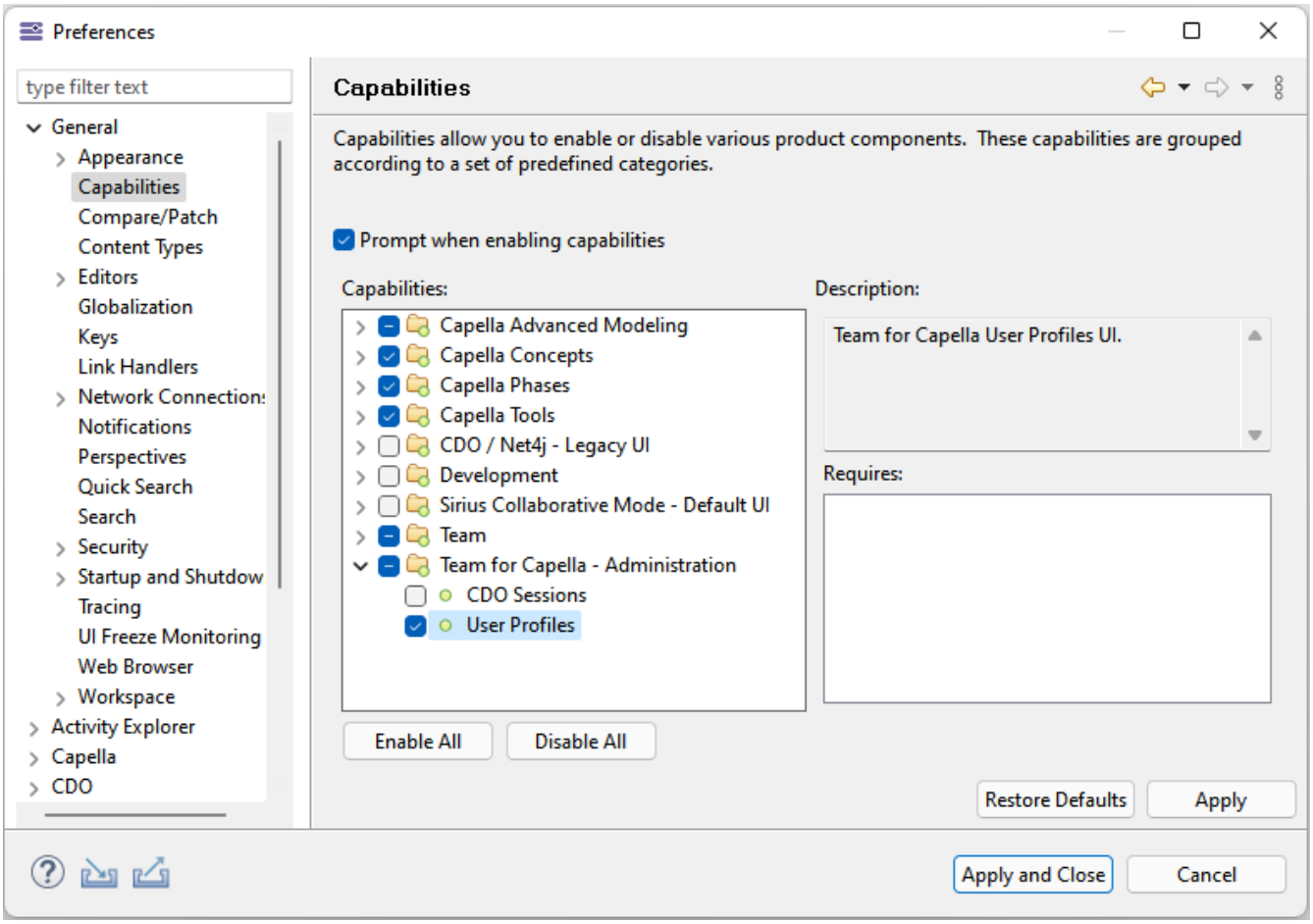
If the server has been started with user profile, the Importer needs to have write access to the whole repository (including the user profiles model). See Resource permission pattern examples section.

If this recommendation is not followed, the Importer might not be able to correctly prepare the model (proxies and dangling references cleaning, ...). This may lead to a failed import.

To use the User Profiles feature in T4C, you first need to install the associated **Team for Capella User Profiles UI** feature from the Team for Capella update site.

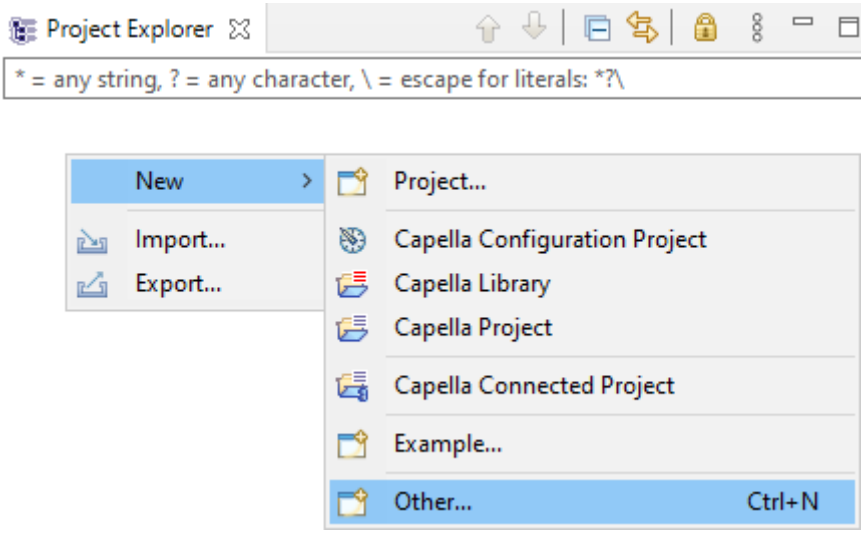


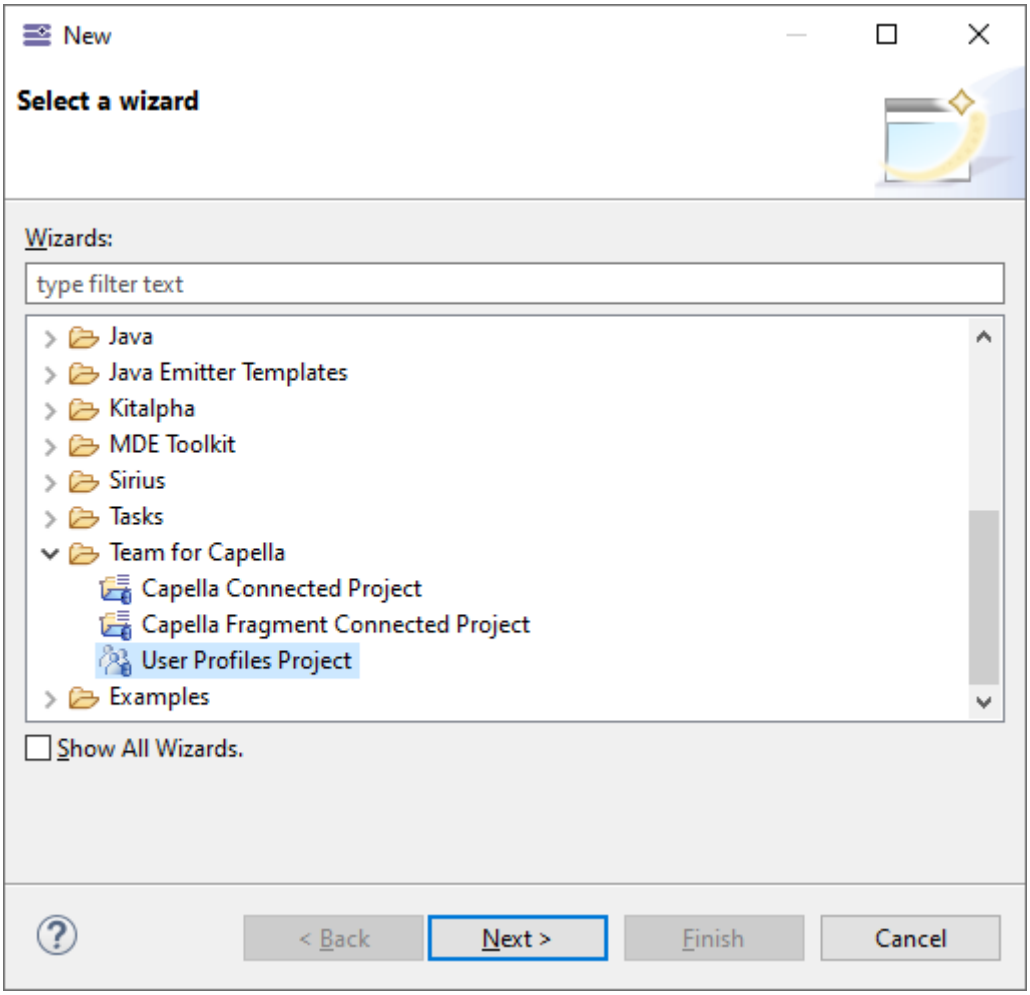
After restarting your T4C client, go to Preferences > General > Capabilities to enable the User Profiles capability.



Connection to the User Profiles Model

You can connect to the user profiles model of a repository thanks to the dedicated wizard:

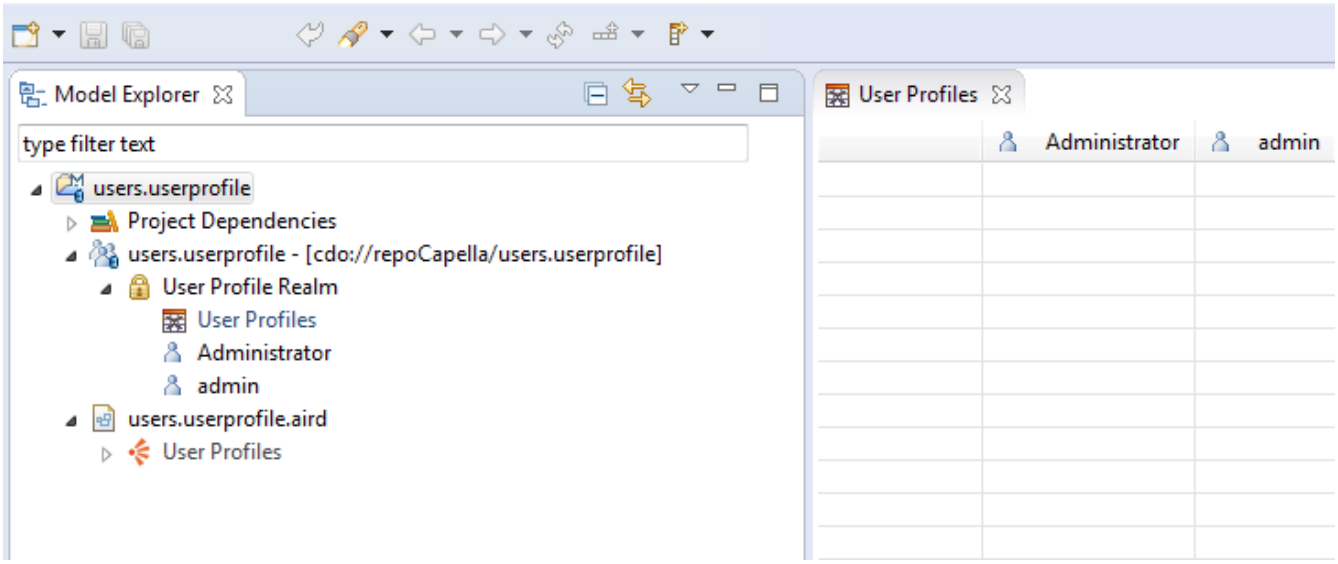




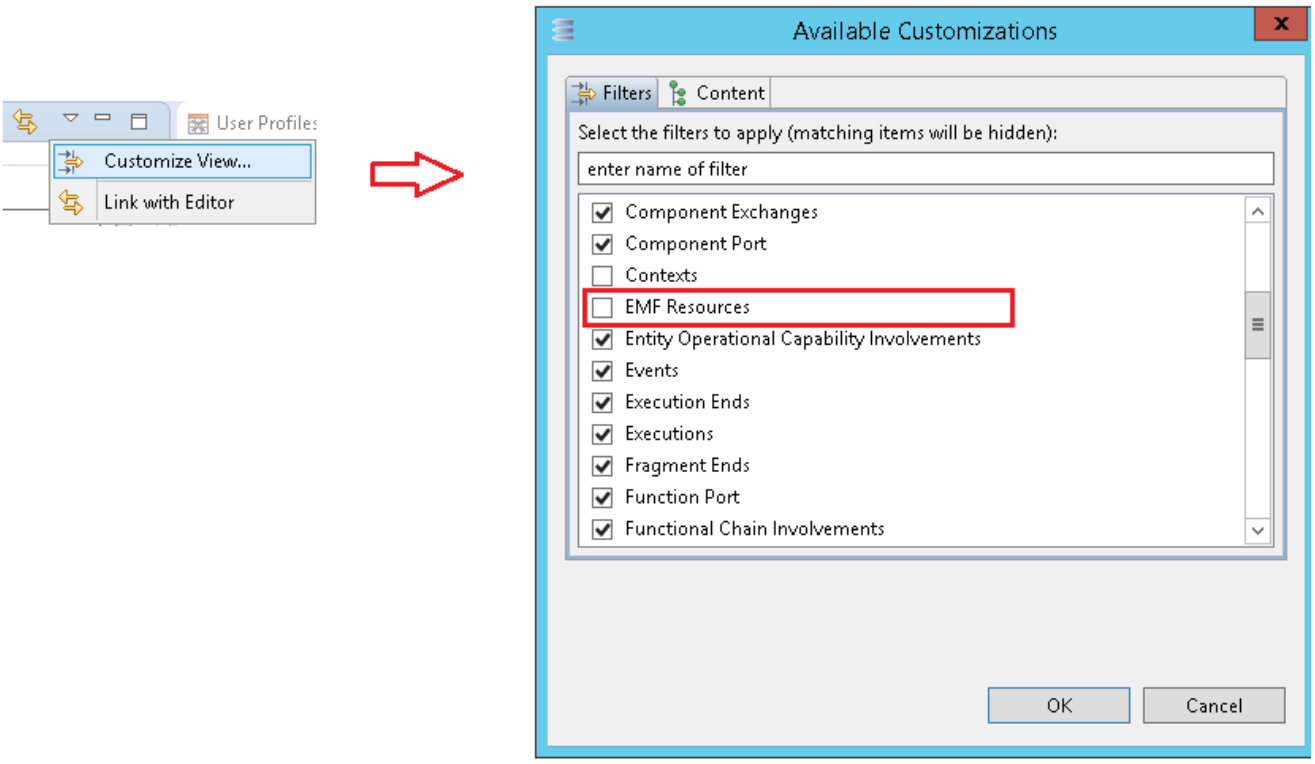
The accounts created by default in the user profiles model are those defined in the **administrators file**. Refer to *Server Configuration/User Profile Configuration*

To be able to change the user profiles model, the Administrator account should be used.

Here is the default user profiles model with its table opened:

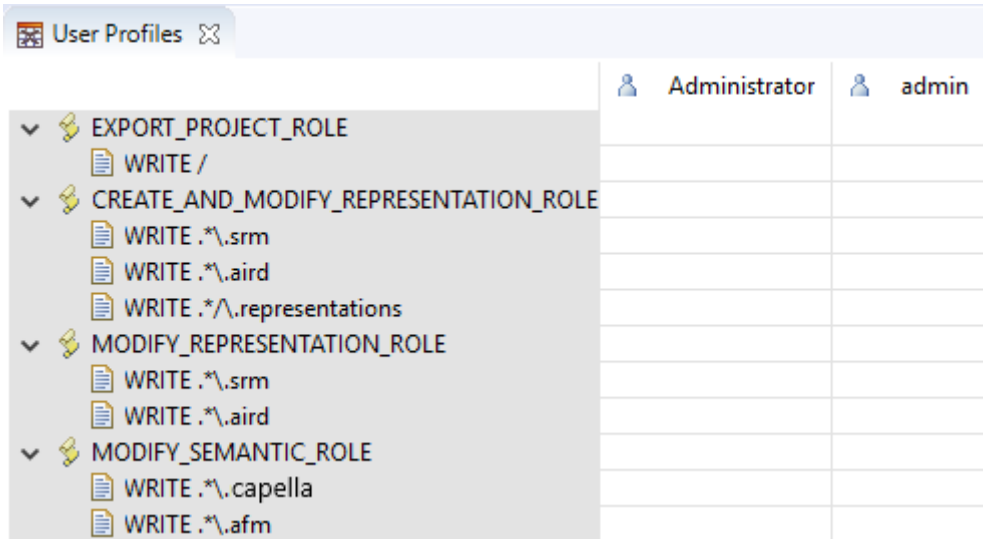


By default, the userprofile resource is hidden. To make it appear under the userprofile project, the EMF Resources filter must be deactivated via the Customize View... dialog.



Default configuration for Team for Capella

When the server is configured with the **User Profiles** functionality, the following roles are automatically created:



These defaults roles are required :

- **EXPORT_PROJECT_ROLE**: is needed to be authorized to export projects. The pattern is only "/" because each project will be exported in the server in a new folder with the name of the project. For exporting projects, the permission to create elements at the root of the repository is therefore needed.
- **CREATE_AND_MODIFY_REPRESENTATION_ROLE**: is needed to be authorized to create and modify representations, but only graphically. This will not allow semantic modifications. This role contains three resource permissions with the following pattern:
  - ".*\.srm", with the lazy loading each representation are placed in a .srm file. This allows

loading only the displayed representations to improve performance.

  - ".*\.aird", this remains the main file aggregating all representations and viewpoints information. Even if the representations are placed in separate files, modifying a representation still updates little information in the .aird file, such as timestamps.
  - ".*/\.representations", with the lazy loading mode, each representation is placed in a folder ".representations" (hidden by default). A permission is therefore needed to create or delete representations in this folder.

- **MODIFY_REPRESENTATION_ROLE**: is needed to be authorized to modify representations but only graphically. This will not allow semantic modifications.
  - The permissions are the same as the previous role, but without the permission on the ".representations" folder to avoid allowing creating and deleting representations.
- **MODIFY_SEMANTIC_ROLE**: is needed to be allowed to modify semantic model elements.
  - The extension files of the semantic resources that are listed as resource permission are provided by the User Profile properties file (by default userprofile-config.properties) referenced by the CDO server configuration file (cdo-server.xml). In this properties file, these file extensions are associated to the "permissions.role.semantic.file.extensions" key and separated by ",".

Installing some plugins introduces new semantic resources that must be considered. These are not automatically added to the associated roles, so you need to create new permissions manually to enable users to modify the new resources. For example, installing Publication for Capella requires the addition of a permission to modify `.traceability` files.

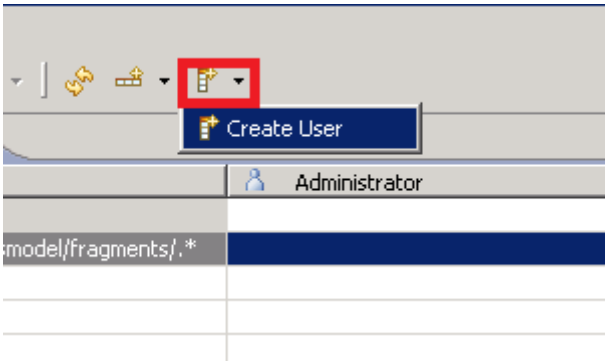
Note that as user created as administrators (in the administrator properties file as presented in the previous part) have full access and do not need to be assigned to any role. Trying to assign roles to administrators will be prevented and a dialog will appear to explain that the administrators already have full access.

Representation Creation/Move Special Case

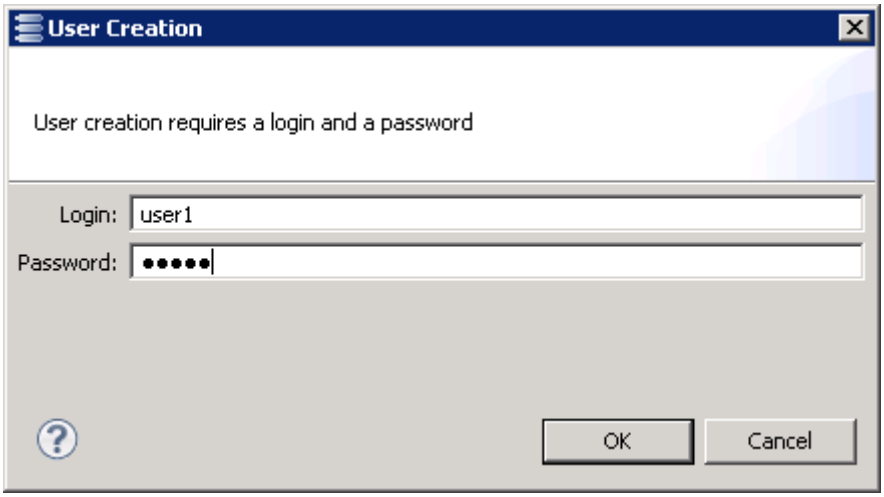
If the user has only a read only right on the semantic element, he cannot create/clone/move a representation on it. If trying, a pop-up will be displayed telling that it failed. More information in Locks and Updates on Diagrams

User Creation

To add a user:

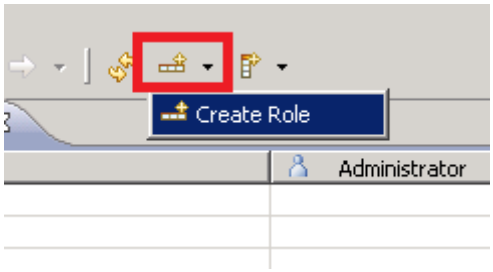


And complete login information



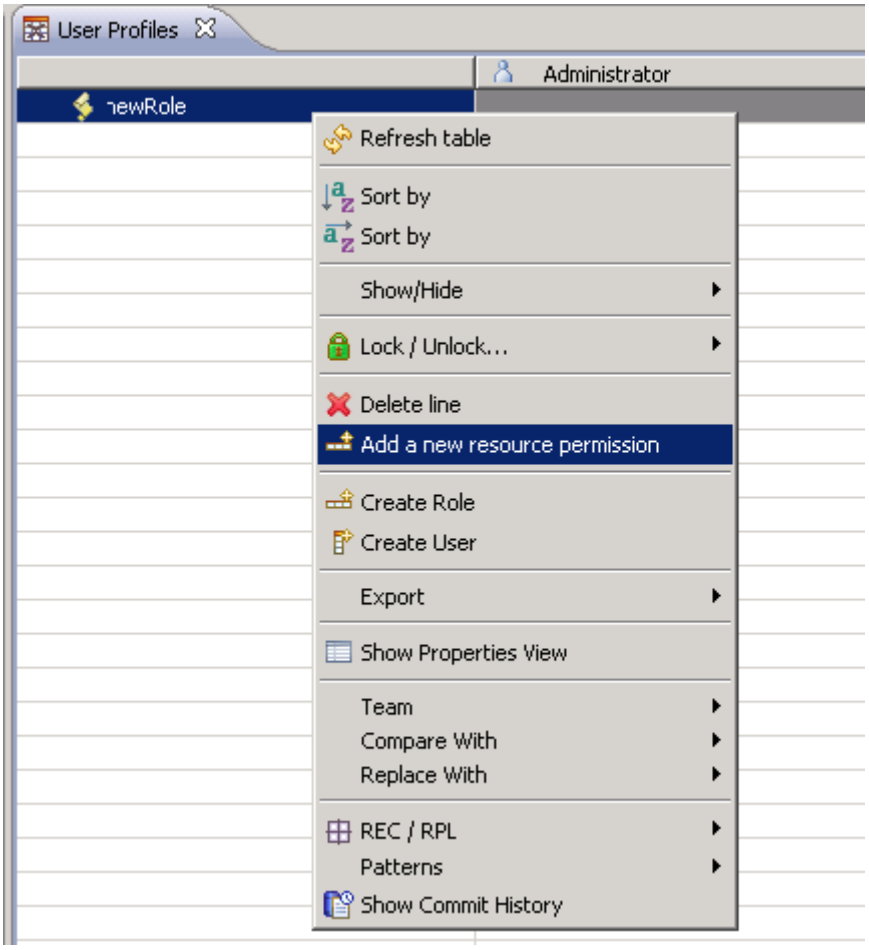
Role Creation and Association with Users

Use the dedicated tool to add a role:

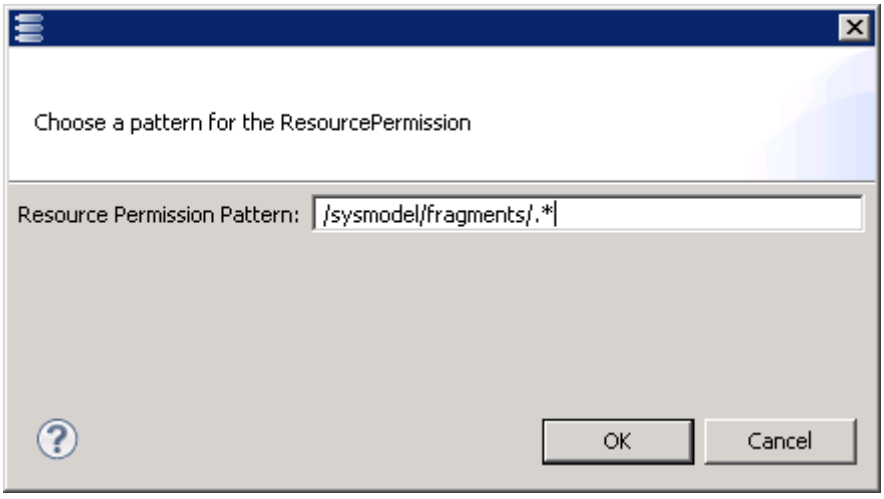


A name can be given to the created role using the Properties view (attribute ID).

Once the new role is created, right-click on it to add resource permission.

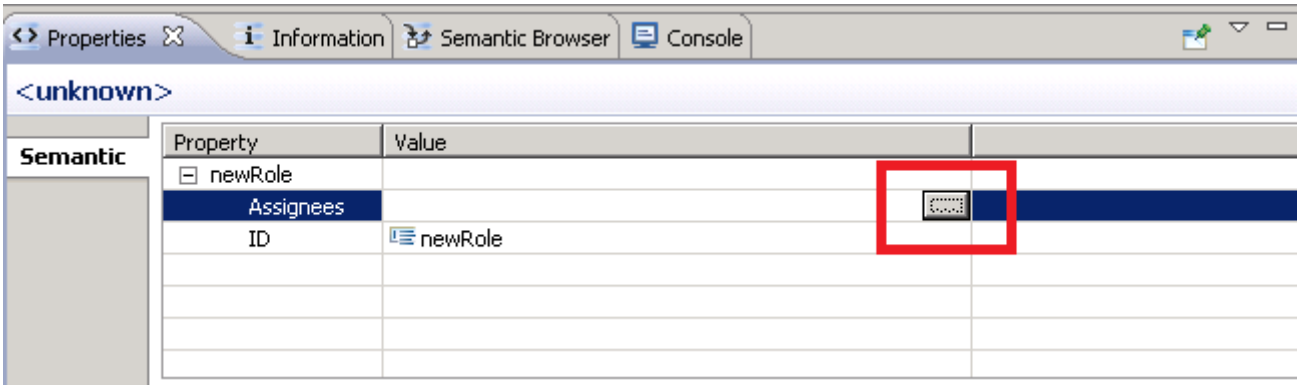


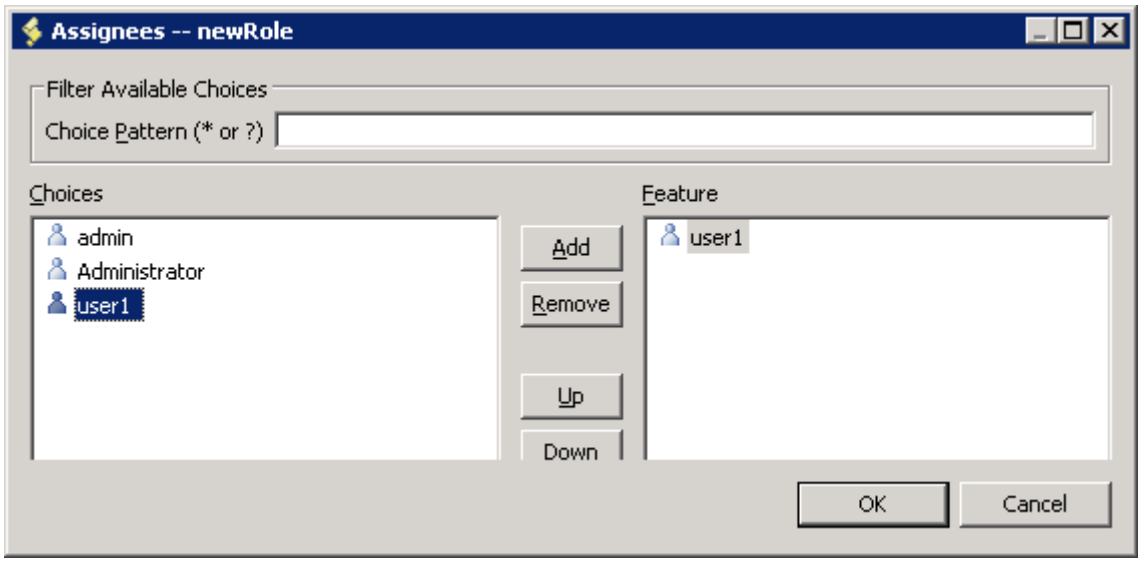
Complete the textbox with path of authorized resource



- " / " represents repository root,
- Resource paths are Java Patterns (https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html),
- Look at the next part to see some pattern examples.

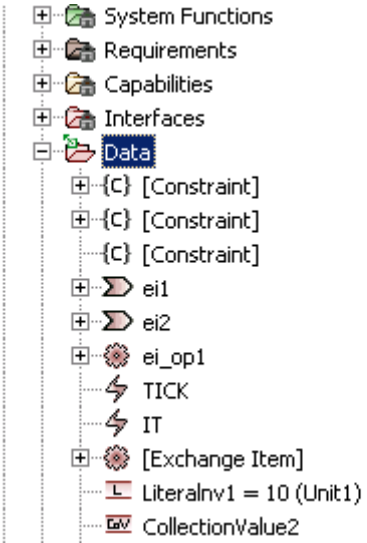
Finally, associate users to a role in the Properties View of the role:





- By default, users have read access to all resources.
- Administrator has a write access on all resources you don't have to assign write permissions for each project for him.
- You can give write or read access to a resource, but empty permission is not supported.
- A user can export a project to a repository only if he has write access on "/".

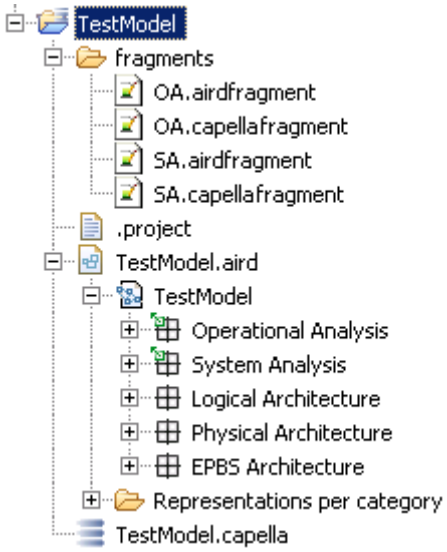
Inaccessible elements for a user have a gray padlock.



Resource Permission Pattern Examples

Since only resource permissions are currently available, to define fine grain permissions on a model, it has to be cut into several fragments.

Here is an example project:



Write access to the whole repository (including the user profiles model)	.* or /.*
Write access to the whole TestModel project	/TestModel/.*
Write access to OA fragments of TestModel	/TestModel/fragments/OA.* or /TestModel/.*OA.*
Write access to OA and SA fragments of TestModel	/TestModel/fragments/(OA|SA).* or /TestModel/.*(OA|SA).*
Write access to the semantic part of TestModel	/TestModel/.*(capella|melodyfragment)
Write access to the representation part of TestModel (diagrams and tables)	/TestModel/.*(aird|airdfragment|srm)
Write access to TestModel but not its fragments	/TestModel/.*(aird|capella|srm) or /TestModel/[^/]*

When dealing with aird and airfragment files, remember to give the **same rights** to srm files (files used to store the representations data when the lazy loading is enabled, the lazy loading is enabled by default).

Note that the project name in a resource permission pattern must be the name coming from the server repository. This is not necessarily the same name as the locally imported project (e.g., if TestModel.team is the name of the locally imported project, putting TestModel.team in the permission pattern will not work).

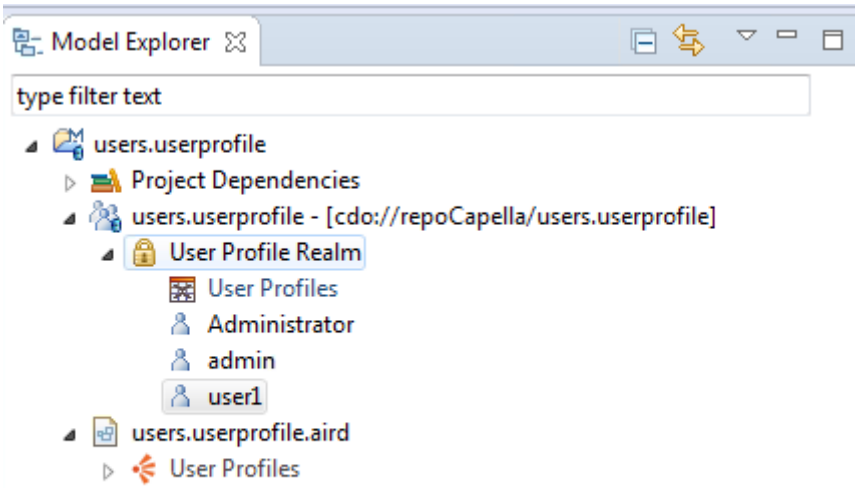
By giving a user write access to a whole project, the user will not be able to create new resources, such as creating the first image on a shared model, or initializing a `traceability` model for Publication for Capella.

Promote a User to Super User

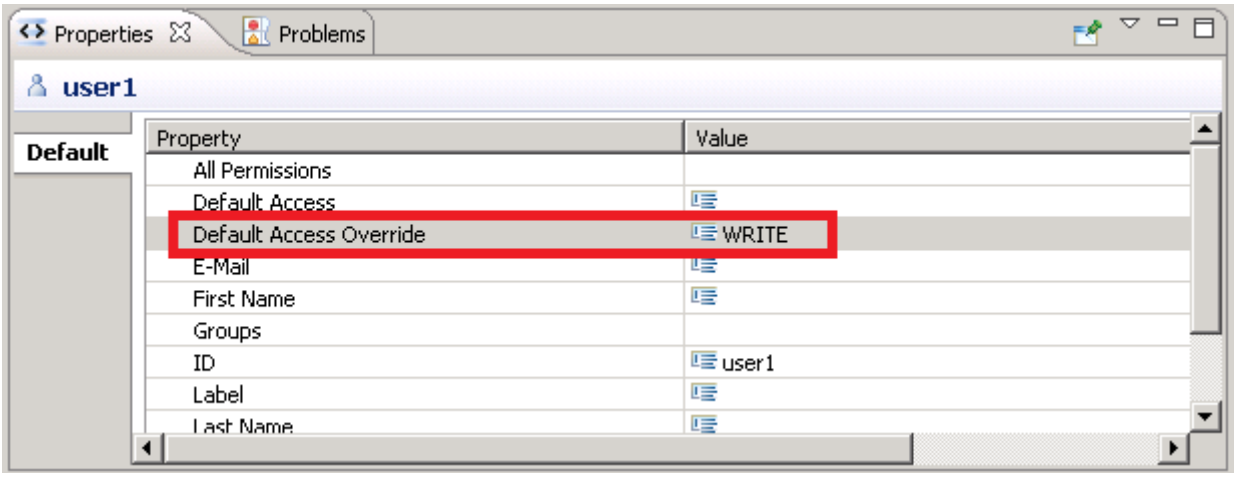
At startup, there is only one superuser: Administrator.

A basic user can be promoted to super user. To do that:

- Connect to the user profiles model,
- Switch to the "Modeling" perspective:
  - Open the "Open Perspective" dialog by clicking on Window > Open Perspective > Other ...
  - Select the "Modeling" perspective.
- Select an account in the "Model Explorer":



- Set the "Default Access Override" to WRITE:



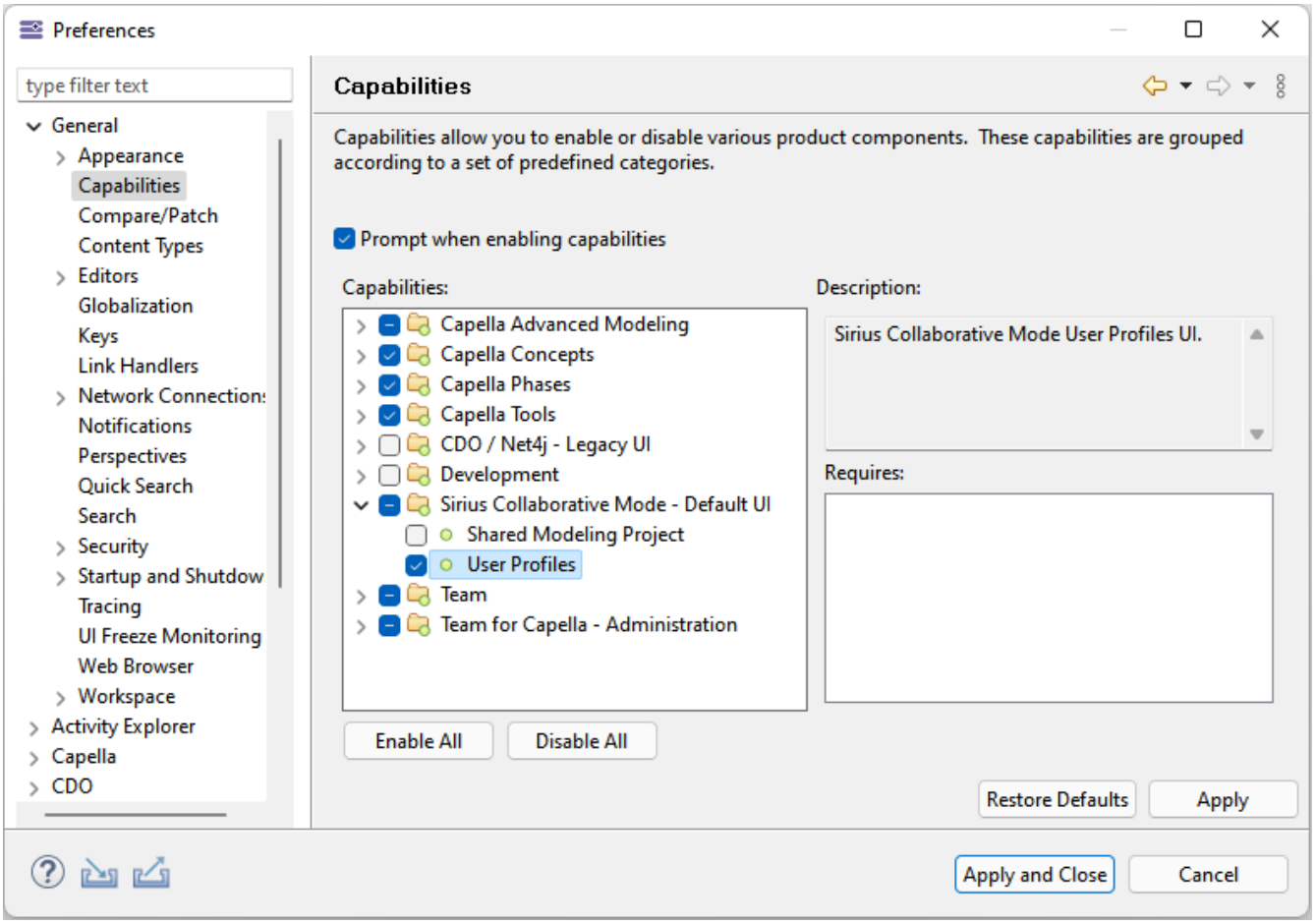
- Save.

Import/Export User Profiles Model

You have the possibility to import a user profiles model; this is the same mechanism as for a Capella project.

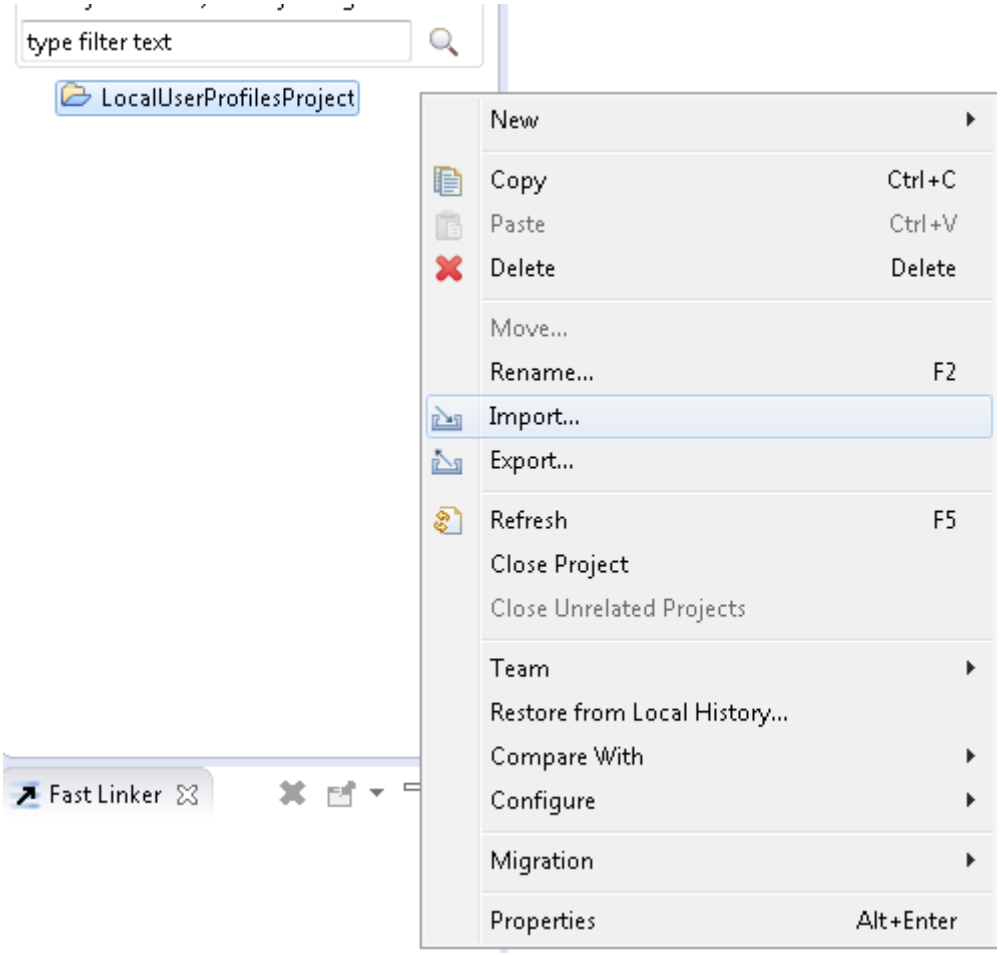
In Team for Capella, you need to enable the **Sirius Collaborative Mode – Default UI** > **User**

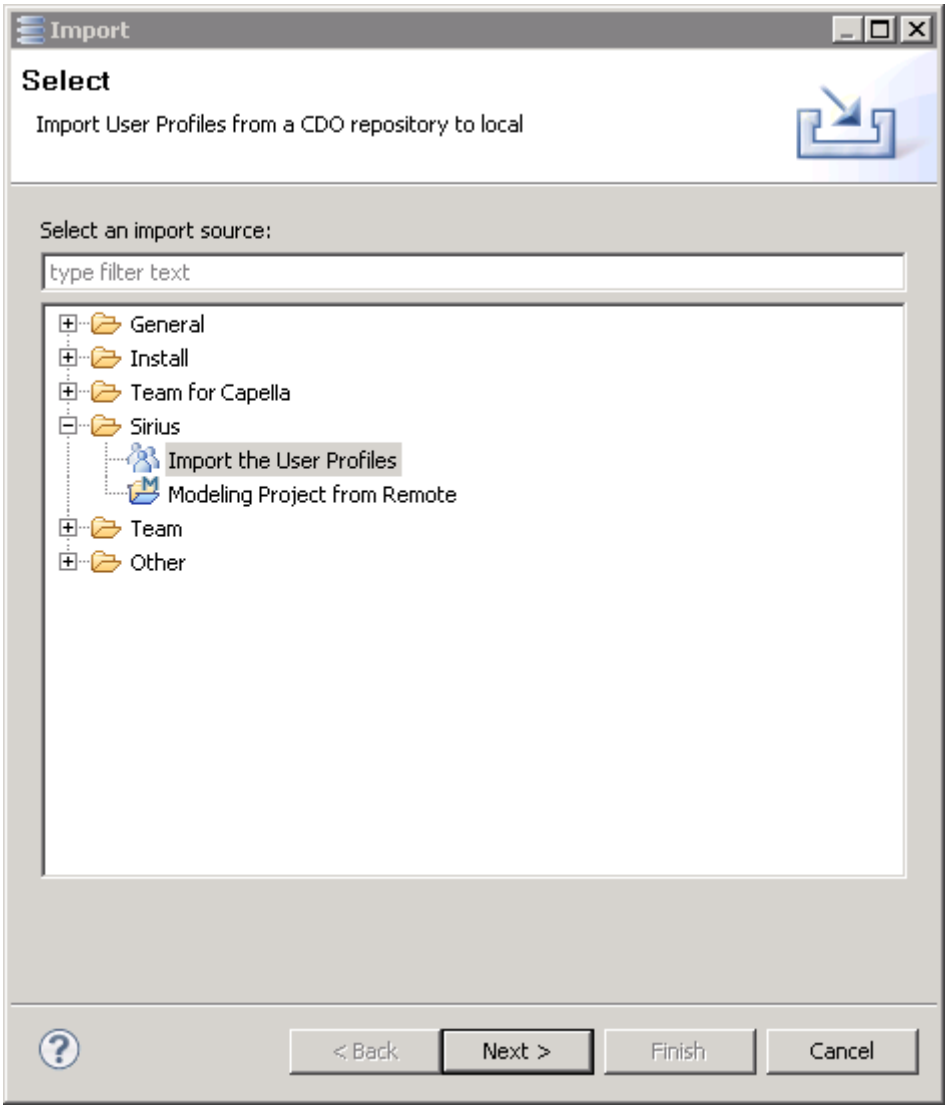
Profiles capability to access the import/export User Profiles functionalities.

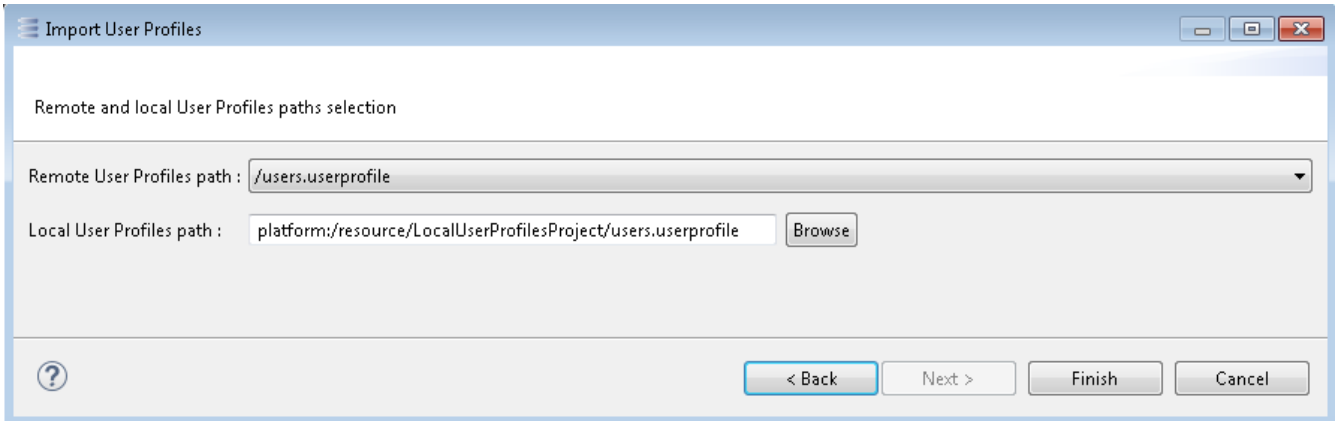


Then, you need to create a general project which will contain the imported User Profile model.

Import User Profiles model:


type filter text
LocalUserProfilesProject
New
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Move...
Rename... F2
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Team
Restore from Local History...
Compare With
Configure
Migration
Properties Alt+Enter
Fast Linker

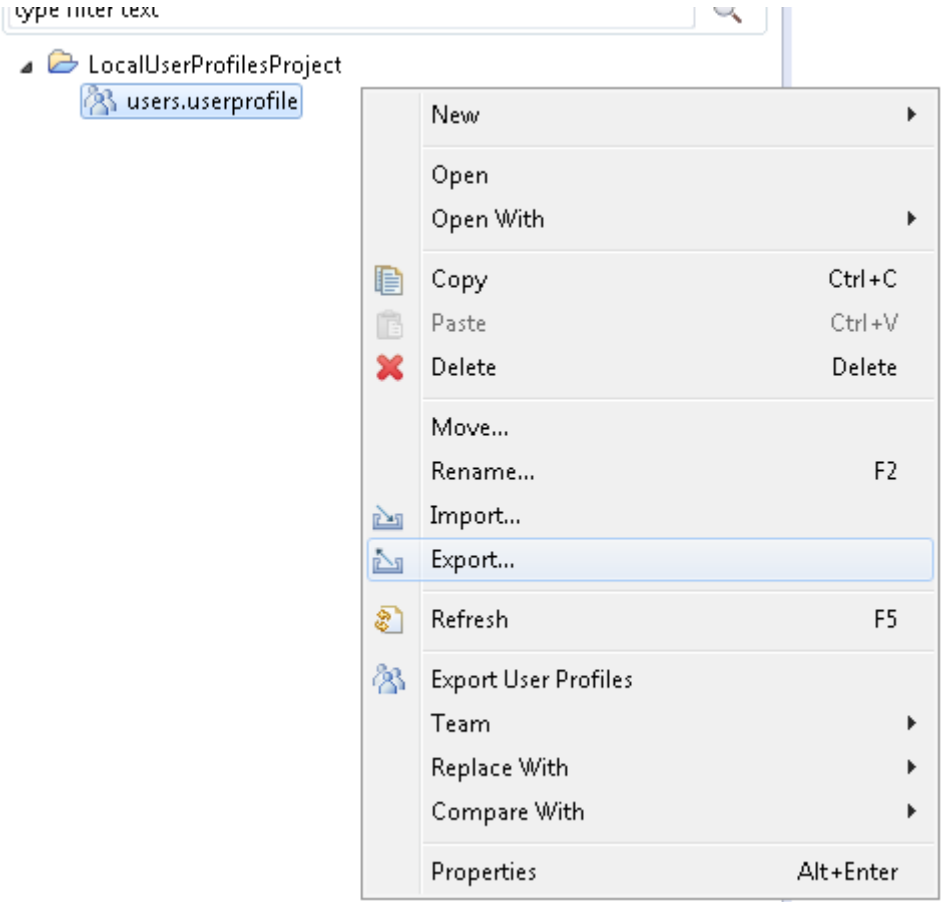


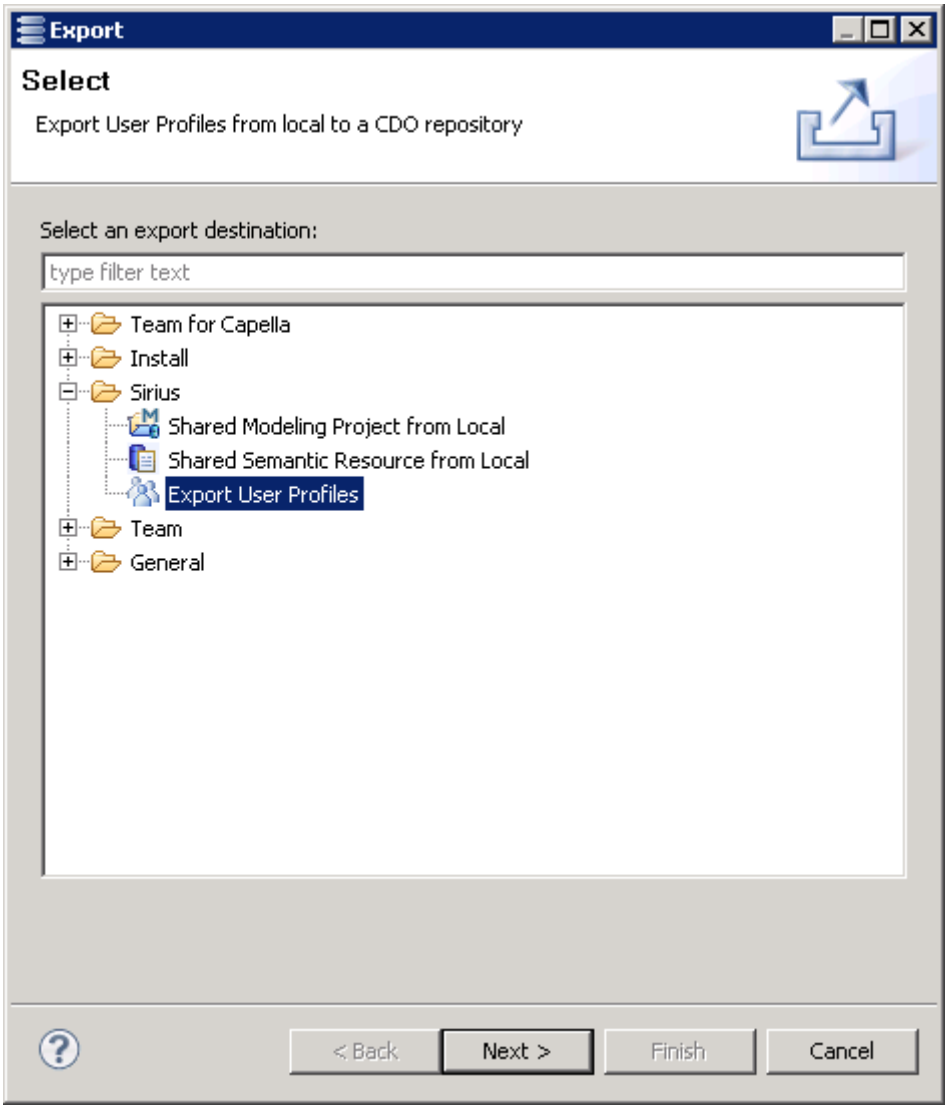


Enter a local URI starting with **platform:/resource/**

Example: **platform:/resource/LocalUserProfilesProject/users.userprofile**

To export, we can create a general project (or reuse the general project created earlier) and put a User Profile model into it, then right-click on the User Profile model and choose Export:


type filter text
LocalUserProfilesProject
users.userprofile
New
Open
Open With
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Move...
Rename... F2
Import...
Export...
Refresh F5
Export User Profiles
Team
Replace With
Compare With
Properties Alt+Enter



How to reuse the user profiles model

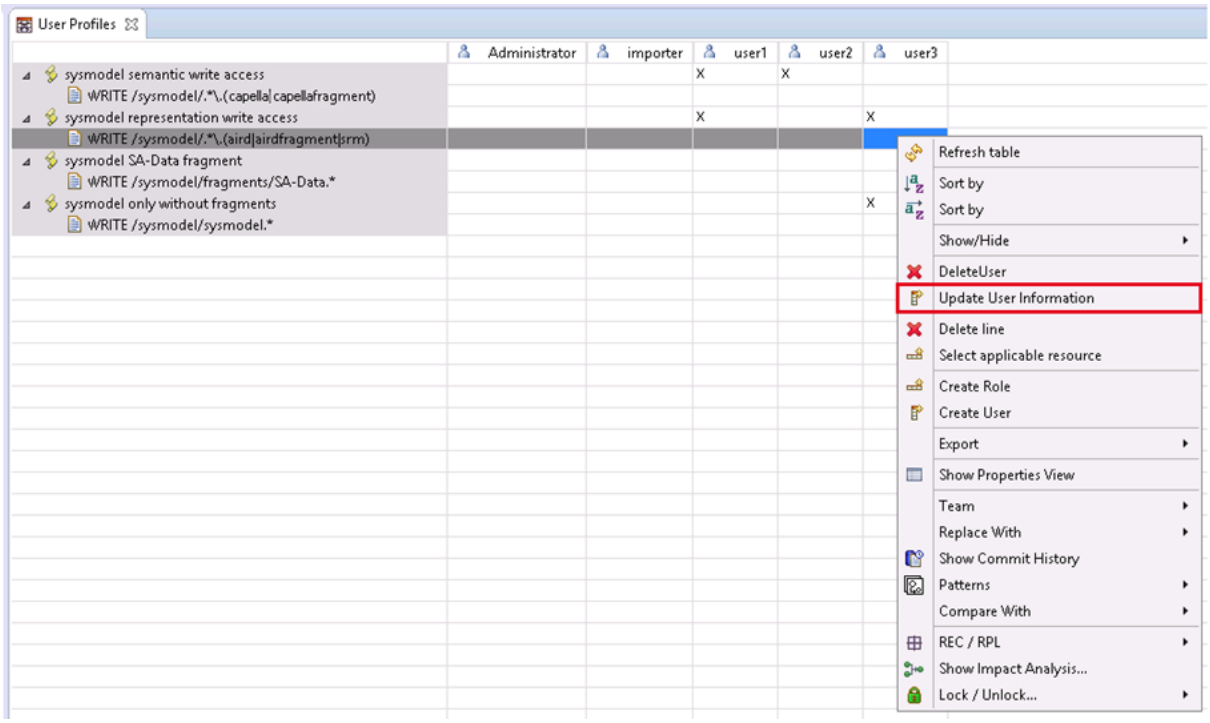
It is recommended that you back up your user profiles model (Refer to *Server Administration/Team for Capella Scheduler/Import user profiles model*).

- You can reuse the user profiles model using the export wizard. You can export it to another repository of either the same server or another server
- In case of DB crash, start your server in standard configuration (Refer to *Server Configuration/Not Authenticated Configuration*), with a clean database. That configuration will not initialize the user profile model. Then export the user profiles model to the CDO repository. Now you can restart the server with user profile; as the user profile model is found, it will not be reinitialized.
- The user profile model can be reused from a Team for Capella version to another. It does not need to be migrated.

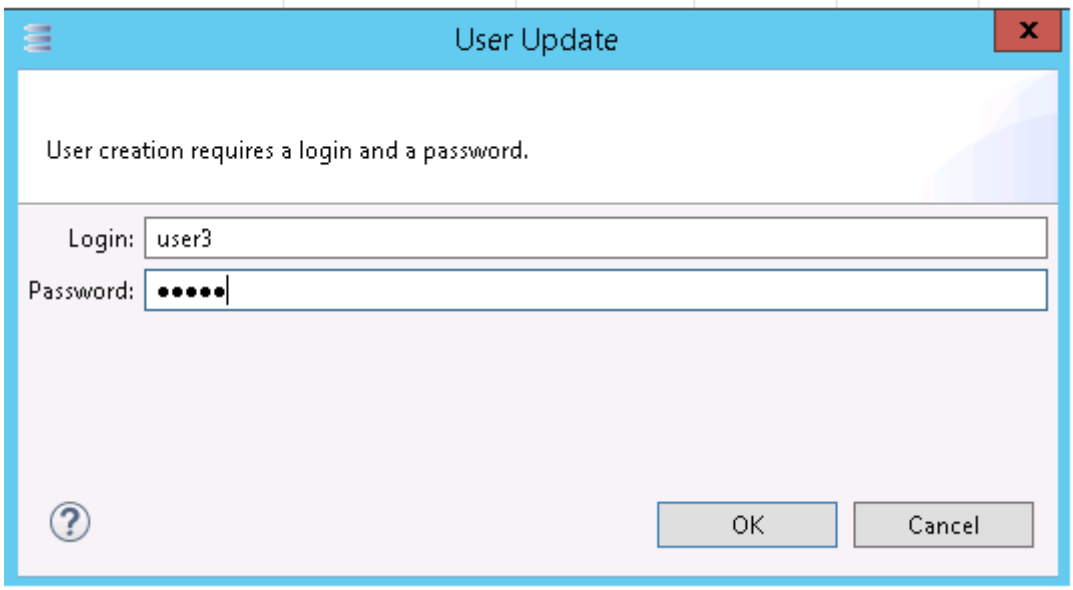
How to change user login/password

User login/password can be modified via the **Update User Information** contextual menu. This contextual menu can be accessed by right-clicking on the column corresponding to the user being modified. Note that this action is done only by right-clicking on one of the cells of the column,

clicking elsewhere (e.g. on the column title) should be avoided.



Once the **User Update** dialog appears, we can modify either user login or password.



- A user cannot modify its own login (the field is read-only).
- If the server is using an external system for authentication (like LDAP), the password field will be hidden as it is not managed by the server.

Troubleshooting

Administrator Password Forgotten

If the administrator password has been forgotten, it will no more be possible to change the user profiles model or export a model to the server.

To give a new password to the Administrator account:

- Stop the server,
- Edit the cdo-server.xml file and comment the line **<securityManager type="collab" realmPath="userprofile-config.properties"/>**. This will deactivate the secured access,
- Start the server,
- Connect to the user profiles model (no password is required),
- Change the Administrator's password,
- Stop the server,
- Uncomment the **securityManager** line,
- Start the server.

Known issues

Please notice the following known issues:

Re-connection to a user profiles model raises error

Chapter 6. Developer Guide

Contents

6.1. Overview

Team for Capella is a collaborative MBSE tool and methodology that relies on the Sirius framework. Both provides extension points and APIs allowing developers to customize and extend Team for Capella. Some of these developments are available as open source add-ons. This documentation will reference some pointers to get started:

- Team for Capella development guidelines
  - As working with shared models have some specific tricks to know, this document lists some recommended guidelines.
- Sirius documentation
  - As Capella relies on Sirius for the representation display and specification, the Sirius documentation is quickly a must read for a developper wanting to provide new representations or viewpoint or extend them.
- Sirius tutorials
  - Tutorials presenting the creation of viewpoint specification projects with Sirius are also a good start for developers.

6.2. Developer Guidelines

To avoid performance issues, some guidelines must be followed.

Viewpoint Generation

It is recommended to generate viewpoint with CDO Native.

Please refer to the Capella Studio Documentation to see how to generate this part of the Viewpoint.

CDO Native Vs CDO Legacy mode

Viewpoints (as described in Capella Guide > User Manual > Overview > Capella Ecosystem) must be generated for CDO.

Nevertheless, if you decide to use the Legacy mode, you can enable it by setting the non-UI preference CDOSiriusPreferenceKeys.PREF_SUPPORT_LEGACY_MODE to true, even it is not a recommended nor supported mode in Team for Capella. For more information, refer to the Activate Legacy mode support.

Diagram extensions

Mapping accesses

Repeated calls to the following methods must be avoided:

- *org.eclipse.sirius.viewpoint.DRepresentationElement.getMapping()* and concrete equivalents: *org.eclipse.sirius.diagram.DDiagramElement.getDiagramElementMapping()* and *getActualMapping()*, *org.eclipse.sirius.table.metamodel.table.DTableElement.getTableElementMapping()*, *org.eclipse.sirius.tree.DTreeElement.getTreeElementMapping()*
- *org.eclipse.sirius.viewpoint.Style.getDescription()*
- *org.eclipse.sirius.diagram.DDiagram.getDescription()*, *org.eclipse.sirius.table.metamodel.table.DTable.getDescription()* and *org.eclipse.sirius.tree.DTree.getDescription()*

For remote models, these methods do not simply access to a reference as the target objects are not shared, then it is recommended to use local variable instead of multiple occurrences of those calls.

Interpreter access

Repeated calls to *org.eclipse.sirius.tools.api.interpreter.InterpreterRegistry.getInterpreter(object)* must be avoided. Note that the *IInterpreter* is the same for the whole *ResourceSet* and corresponding Sirius *Session*. If you already have this *Session*, you can use *org.eclipse.sirius.business.api.session.Session.getInterpreter()*.

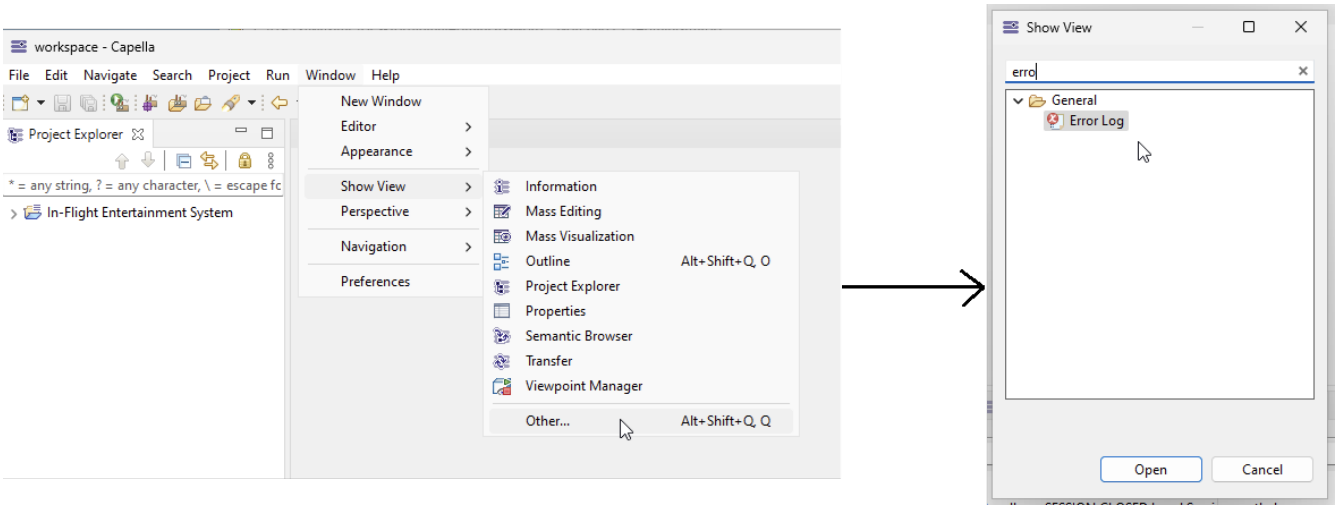
Chapter 7. Troubleshooting Guide

This chapter provides information on resolving issues you might encounter when running Team for Capella. This section is suitable for all profiles.

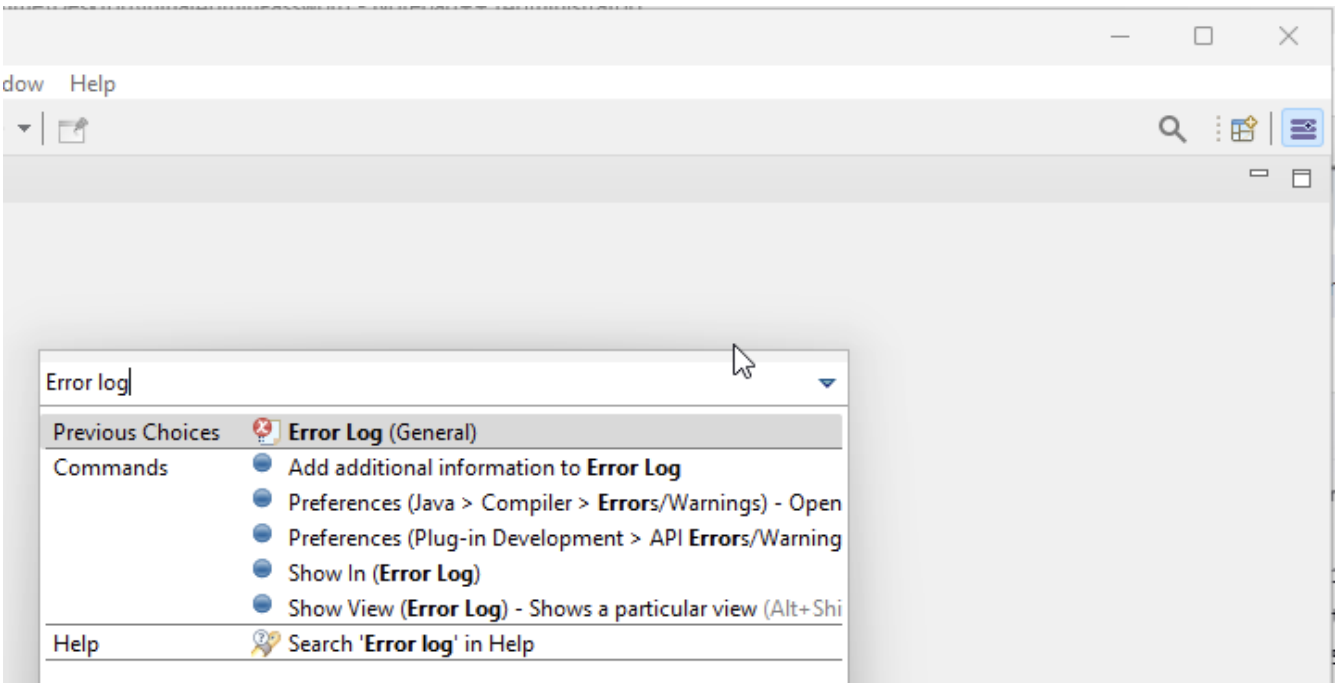
The Error Log View

To open the **Error log** view:

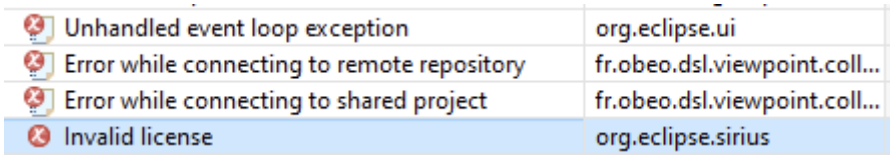
- Window > Show View > Other..., then search `Error Log`:



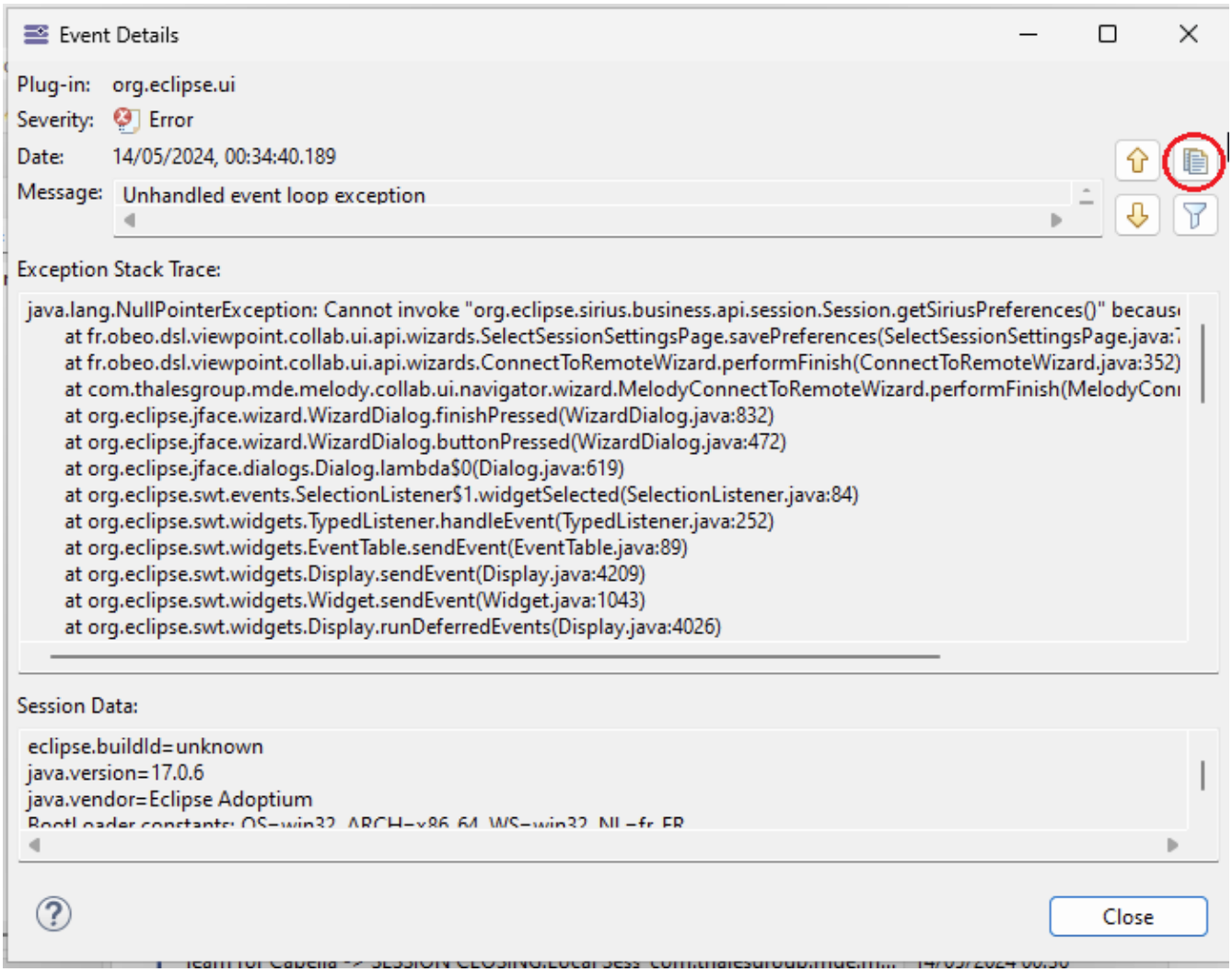
- Or, click on the search icon and search `Error Log`:



When there's a problem, the first thing to check is the error log:

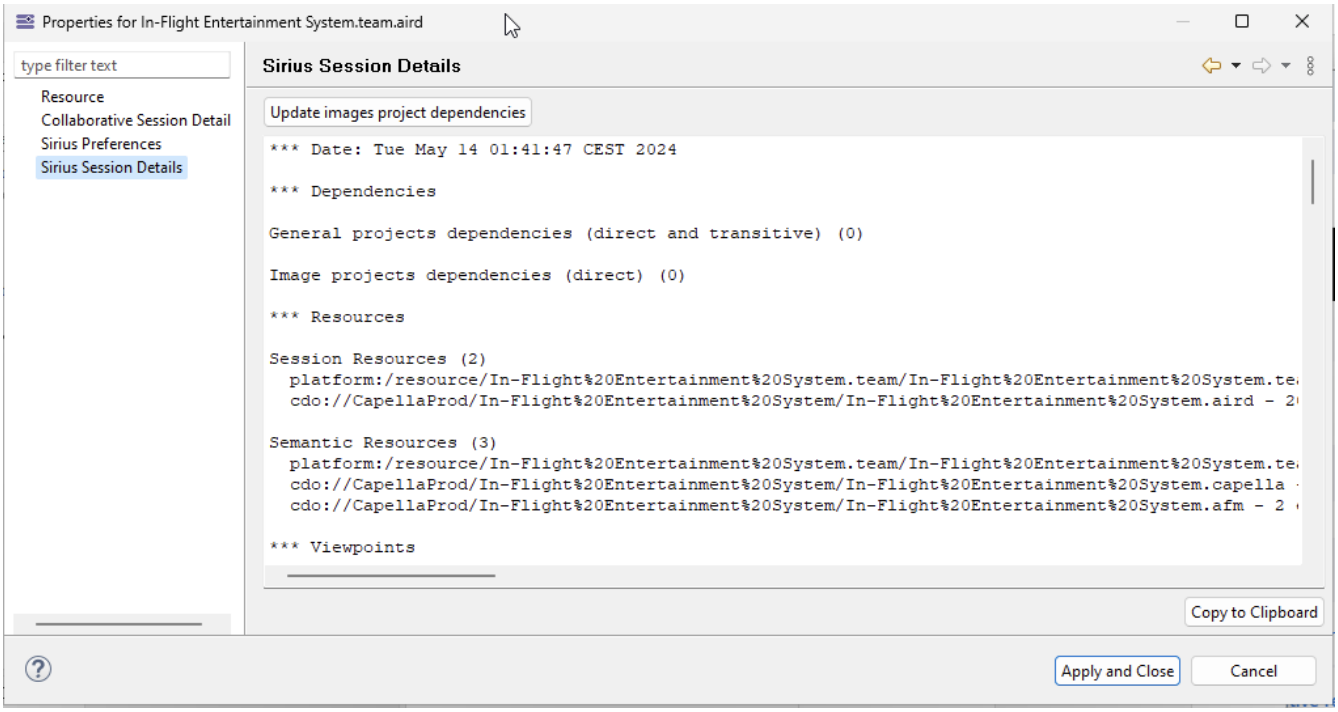


If errors are present, you can view the call stack and copy it:



Sirius Session Details

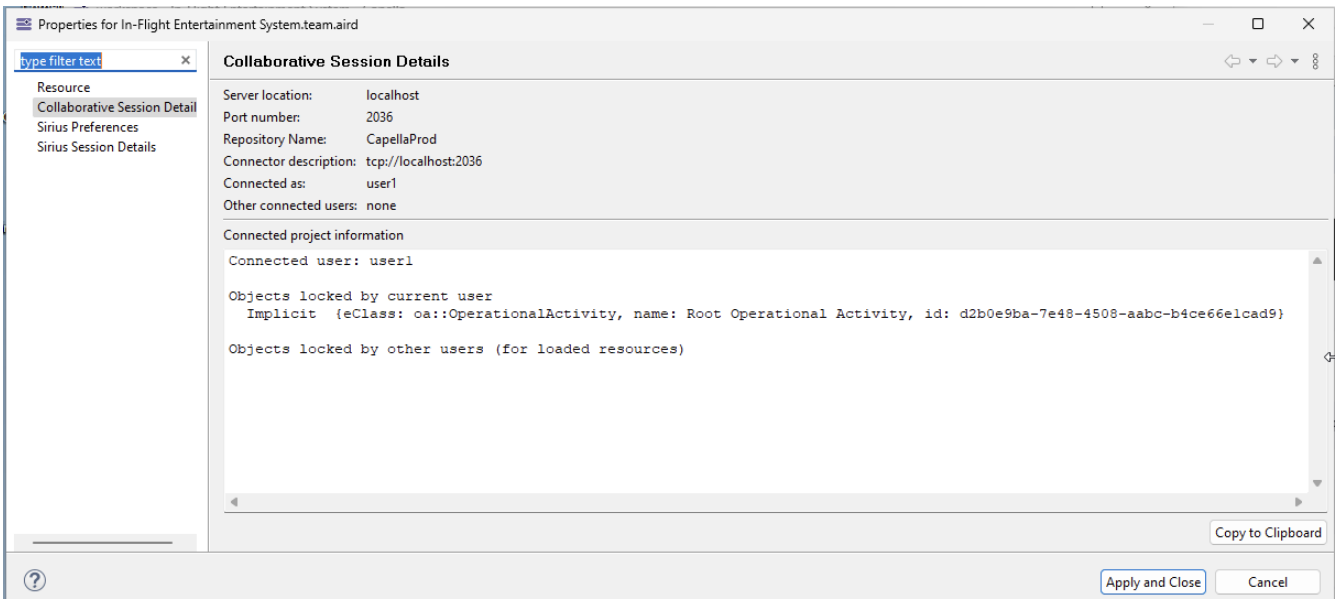
The Sirius Session Details is available in properties page of the `.aird` file of a Capella project with opened session (right-click > Properties):



This page contains information about resources, size, number of elements, number of representations, activated Viewpoints.

Collaborative Session Details

The Collaborative Session Details is available in properties page of the `.aird` file of a Capella project with opened session (right-click > Properties):



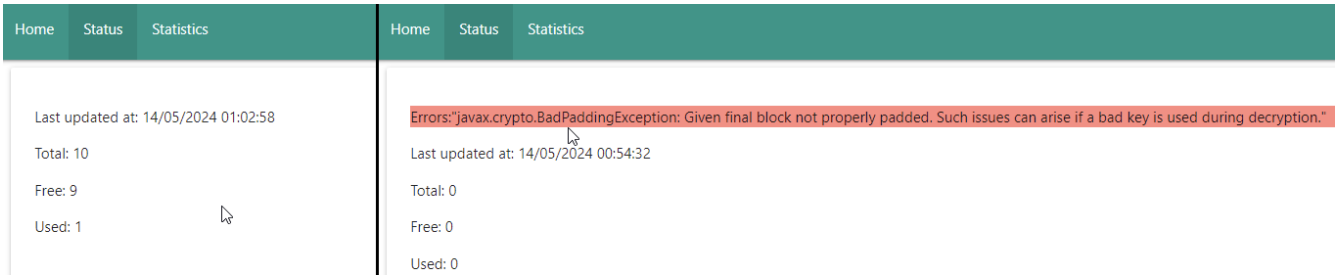
This page contains information about user, locked objects, connection.

Invalid License

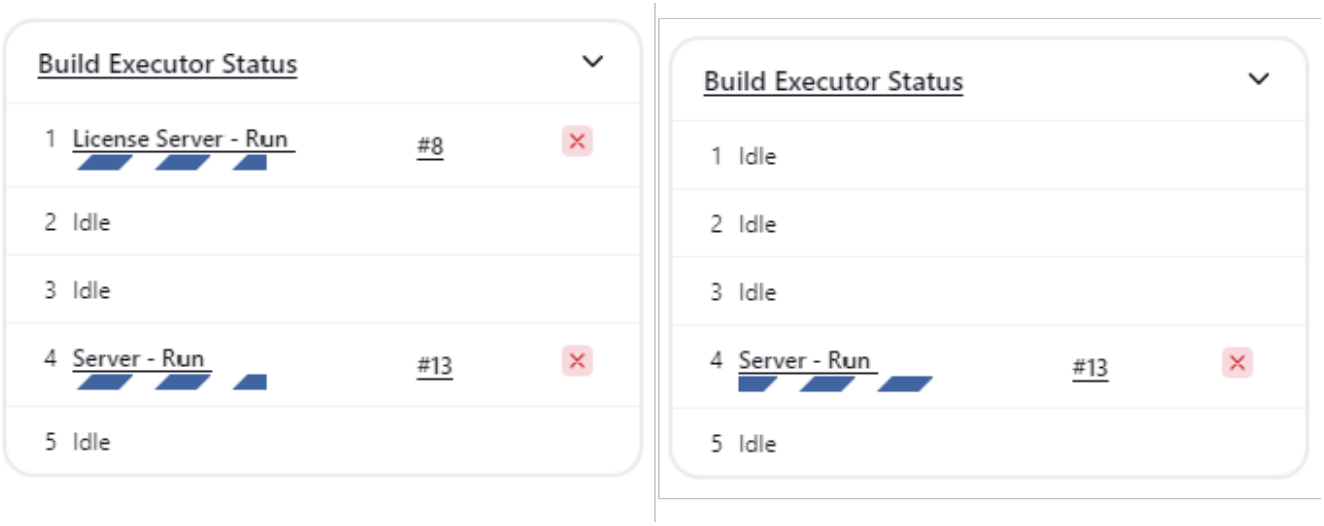
When there’s an "invalid license" issue, you should check:

- The license server monitoring, in Status and Statistics tabs.For example, on the left you’ll see the

status tab with no error, and on the right with an error:



- The `License Server - Run` job status and log, at the left the License Server is running but not at the right:



If a `BadPaddingException` is reported like here in the log:

```
!ENTRY fr.obeo.oo15oo.oo179oo.oo6083oo.app 1 0 2024-05-14 00:54:32.121
!MESSAGE message from client : IP:127.0.0.1 ID:Maxime|96-58-CF-FB-61-F8

!ENTRY fr.obeo.oo15oo.oo179oo.oo6083oo.app 1 0 2024-05-14 00:54:32.122
!MESSAGE Requesting license for IP:127.0.0.1 ID:Maxime|96-58-CF-FB-61-F8
fr.obeo.oo15oo.oo179oo.oo6083oo.Oo43233oo: javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is
used during decryption.
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41186oo.decode(Oo41186oo.java:872)
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41186oo.getString(Oo41186oo.java:745)
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41186oo.requestLicense(Oo41186oo.java:180)
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41185oo.handleMessage(Oo41185oo.java:351)
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41185oo.run(Oo41185oo.java:282)
        at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)
        at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)
        at java.base/java.lang.Thread.run(Thread.java:833)
Caused by: javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
        at java.base/com.sun.crypto.provider.CipherCore.unpad(CipherCore.java:859)
        at java.base/com.sun.crypto.provider.CipherCore.fillOutputBuffer(CipherCore.java:939)
        at java.base/com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:735)
        at java.base/com.sun.crypto.provider.AESCipher.engineDoFinal(AESCipher.java:436)
        at java.base/javax.crypto.Cipher.doFinal(Cipher.java:2205)
        at fr.obeo.oo15oo.oo179oo.oo6083oo.Oo41186oo.decode(Oo41186oo.java:868)
        ... 7 more

!ENTRY fr.obeo.oo15oo.oo179oo.oo6083oo.app 4 0 2024-05-14 00:54:32.169
!MESSAGE javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.
[ERROR] javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can arise if a bad key is used during decryption.

!ENTRY fr.obeo.oo15oo.oo179oo.oo6083oo.app 1 0 2024-05-14 00:54:32.170
!MESSAGE No token is available for IP:127.0.0.1 ID:Maxime MAC:96-58-CF-FB-61-F8 out of  0 tokens
```

- Stop the `License Server - Run` job

- Delete the `.ols` files in `lic-server/OLS`
- Re-extract the `OLS_xxx.zip` in to have 4 .ols files in `lic-server/OLS`
- Start the `License Server - Run` job
- Try to connect again with the Team for Capella client

Connection Issues

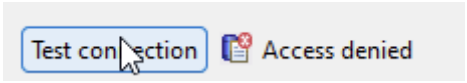


Figure 1. the password is wrong

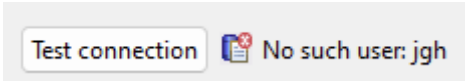


Figure 2. the username is wrong

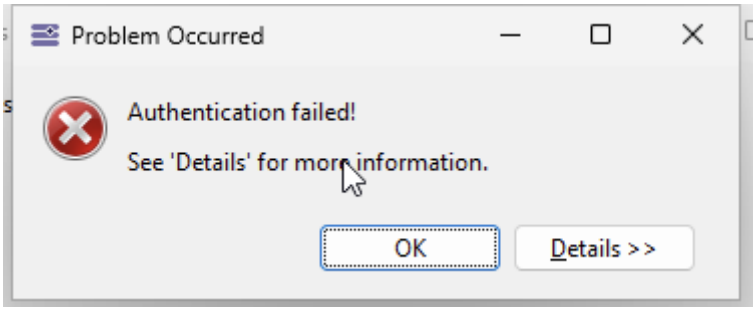


Figure 3. the username or the password is wrong

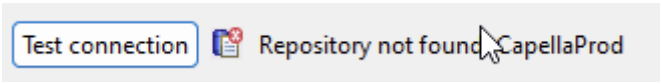


Figure 4. the repository name is wrong or the repository is stopped

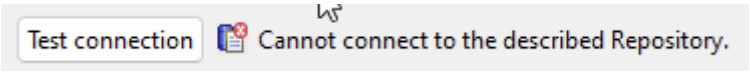


Figure 5. the server is stopped or the hostname/port/connection type is wrong

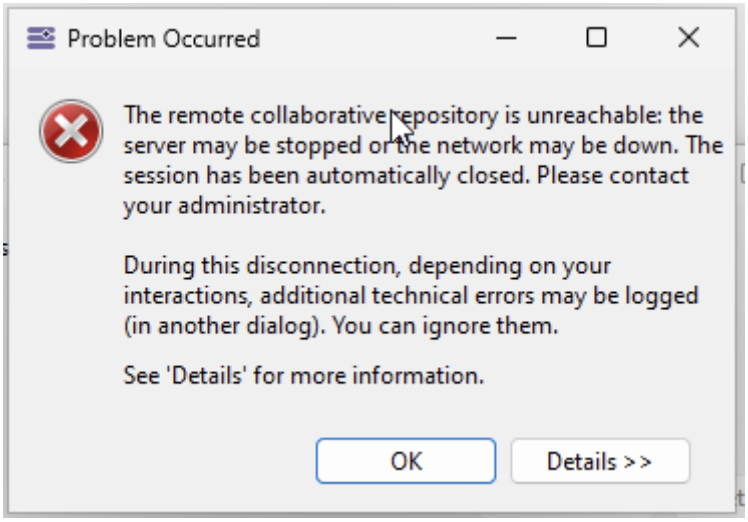


Figure 6. the server is stopped or the hostname/port/connection type is wrong

Stopped Server or Repository

If the server or the repository is stopped:

- Check the `Server - Run` Job status and its console log,
- launch `Server - List active repositories` job and check the console log,
- check the status and logs of the latest `Project  Import  xxx` jobs,
- check the job configuration: some options can stop the repository or the server in the case of issue during backup:
  - `-stopRepositoryOnFailure true` (in default job configuration),
  - `-closeserveronfailure true`,
- analyze the job logs, most common issues:
  - a new add-on has been added in the client bundle without installing it on the server-side.
  - corrupted data (invalid representation descriptor, wrongly managed dangling/proxy reference).

No More Responding Server

If in previous check (Stopped Server or Repository), server is still running, check:

- `Backup  Database` job status and duration
- `Project  Import - xxx` job (targeted repo) status and duration
- Capella/OpenJDK Platform binary processes in the Windows task manager:

Nom	Statut	21% Processeur	43% Mémoire	0% Disque	0% Réseau
capella.exe		0,6%	968,6 Mo	0 Mo/s	0 Mbits/s
OpenJDK Platform binary		1,2%	564,1 Mo	0 Mo/s	0 Mbits/s

- No responding clients, with longer than usual job duration or failed jobs, and high CPU/RAM consumption can indicate that Java `XMX` (and maybe RAM) must be increased for importer, server and/or client.
- Make sure that sum of all `XMX` (`importer.bat`, `server.ini`, `lic-server.ini`, Jenkins) is below the physical RAM amount.

No More Responding Clients on Shared Client VM

If server issue is excluded (No More Responding Server):

- Try to disable “link with editor” capabilities  in Semantic Browser and other views.
- Reduce the number of open editors.

- Check the number of connected users.
- Check the number of launched Capella instance.
- Check the RAM amount of each Capella process in the Windows task manager.
- Check that there is no SWAP.
- Evaluate the xmx increase in `capella.ini`.

Importer: Fail to Start the New CDOServer

If the importer fails with this error:

```
javax.xl.stream.XMLStreamException: ParseError at [row,col]:[1,1]
```

It means the target repository is stopped or does not exist.

Job failing with wrong argument

If a job like "Server - List connected projects and locks" fails with an error message similar to

!MESSAGE port must be an integer java.lang.NumberFormatException: For input string: "-Dosgi.requiredJavaVersion=14" at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:68)

You should check the capella.ini or the exectued script file, as the "-vmargs" argument may have been deleted.

Job failing with parsing issue starting on -vmargs

The -vmargs argument needs to be on its own line without other arguments. This issue has been noticed when users paste their "-DOBEO_LICENSE_SERVER_CONFIGURATION=xxx" argument on the "-vmargs" line, resulting in jobs failing with this kind of log:

```
!MESSAGE Parameter error: Could not parse -vmargs
-DOBEO_LICENSE_SERVER_CONFIGURATION=xxx. It is expected to be followed by an argument
!SUBENTRY 1 fr.obeo.dsl.viewpoint.collab.tools 4 0 2024-07-19 08:32:24.952
!MESSAGE Parameter error: Could not parse -Dosgi.requiredJavaVersion=17. It is
expected to be followed by an argument
!SUBENTRY 1 fr.obeo.dsl.viewpoint.collab.tools 4 0 2024-07-19 08:32:24.952
!MESSAGE Parameter error: Could not parse -Dosgi.dataAreaRequiresExplicitInit=true. It
is expected to be followed by an argument
!SUBENTRY 1 fr.obeo.dsl.viewpoint.collab.tools 4 0 2024-07-19 08:32:24.952
```