



Top Version

Administrator Manual

Software Version 1.3.0

EducateIT GmbH

Phone: +41 32 355 39 00

E-Mail: info@educateit.ch

Address: Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland

Contents

	Preface	5
	Features	5
	Version for this Manual	5
	See Also	5
Chapter 1	Repository	7
	Introduction	9
	File Structure	11
	The repository.xml File	12
	The info.xml File	14
	The Version Sub Directory	14
	Delta Package Format	15
	The delta.xml File Format	15
Chapter 2	Client	17
	Mode of Operation	19
	Check the Current Installed Version	19
	Get the Required Version	20
	Compare Versions	20
	Update Project	20
	Clean Up and Start Program	20
	Configuration	21
	Command Line Arguments	25

Chapter 3	Manager	27
	Creating a New Repository	29
	Create a New Project	29
	Create a New Version	31
	Create a Channel	32
	Save the Repository	32
	Features and Preferences	33
	Warn Sign on Changed Version	33
	Preferences	33
	Creating Reverse Deltas	36
Chapter 4	Appendix	37

Preface

The *Top Version* system simplifies frequent updates of a software package called “project” in this manual. Speed and robustness were the goal while designing this system.

There is a client which can be shared to update many different projects on a computer, or it can be bundled with the project itself and delivered as automatic update solution.

A repository manager simplifies the task of maintaining the different versions of a project, and automatically created the necessary delta packages which are required for a fast update.

This manual is split into different chapters, describing the configuration of the client, the usage of the repository manager and the structure and format of the repository.

Features

- Fast but robust updating for projects.
- Delta update which only updating changed, added or removed files in the project.
- Flags to enforce a complete update of a project.
- Different channels to use multiple versions of the same project. For example a production and a beta version.
- Open XML and ZIP based repository format with XML schema to verify the XML files.
- Easy to use repository manager with graphical interface.

Version for this Manual

The version of this manual is bound to the version of the Top Version client. It describes version 1.3.0 of the client, and version 1.0.2 of the repository manager.

See Also

The client is using the *Configuration B* system for the configuration. To fully understand all aspects of the configuration, it is recommended to read the document “*Configuration B - Administrator Manual*”. You should get this document together with this manual. If not, please request it from our company.

Chapter 1

Repository

This chapter describes the format and structure of a repository. Regardless the structure is created and managed by the repository manager, you should know and understand the format.

Introduction

There are some terms which are used in the *Top Version* system to describe various parts. One of this terms is *repository*.

The *repository* stores any number of different *projects*. In fact, the repository is a defined structure of files and folders. This structure can be created at any place, usually on a shared place in the network with appropriate access rights.

A *project* in this *repository* is a structure of files, for example a software installation, which is maintained on a target system. From this file structure, various versions can exist in parallel. Each version has a unique name and a given position in the *version vector*.

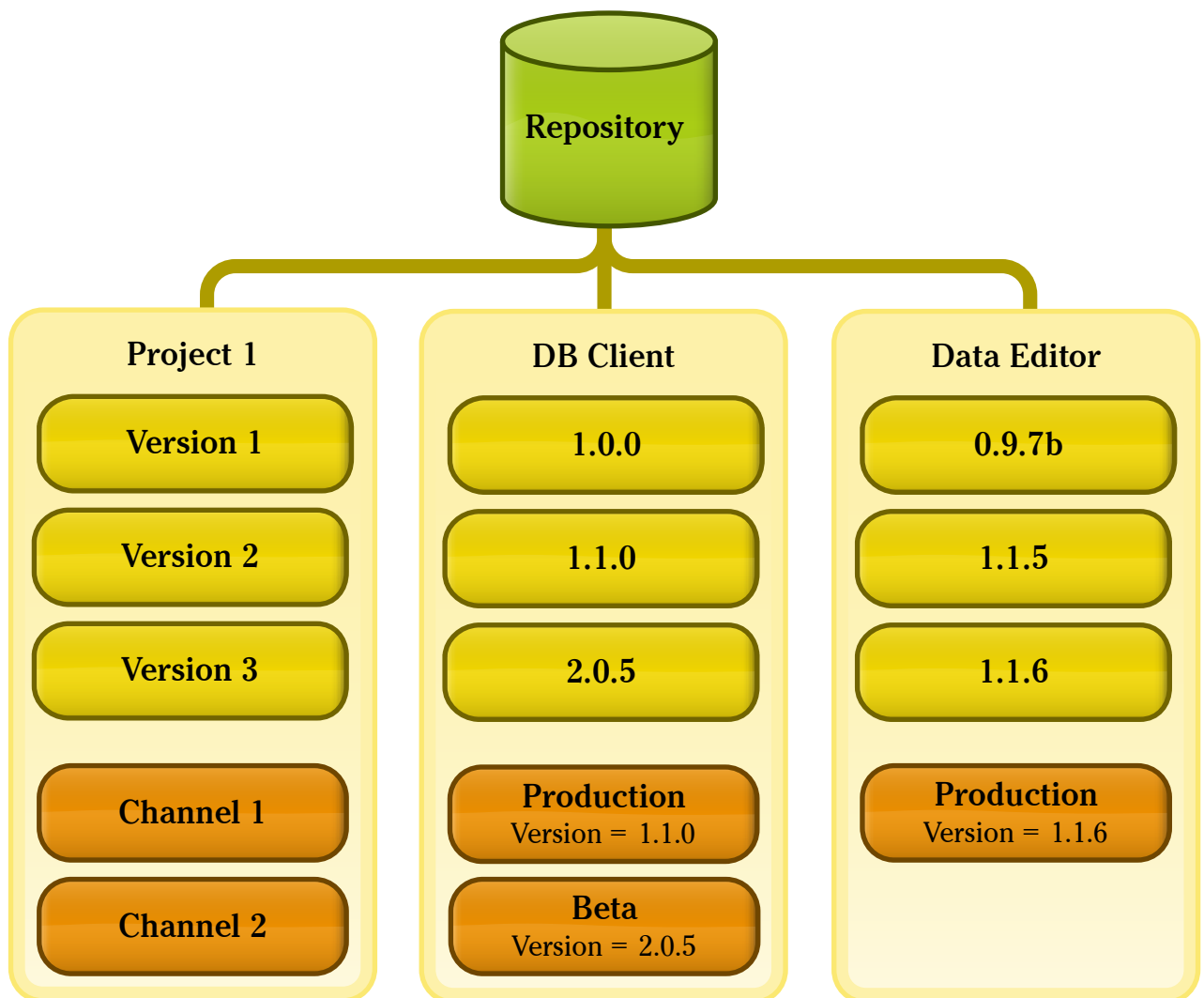


Illustration 1 The used terms and how they are related to the repository.

The position of the *version* is about the time line. *Top Version* uses the version names just as identifiers, but makes no attempt to interpret them in any way. You have to tell the system the order of the versions. Which version is newer or older than another.

Each project can have one or more *channels*. A *channel* defines the current version which is in use. You probably will create a “Production” channel. This channel always defines which version shall be used on productive systems. But you can create more channels, probably a channel called “Testing” which defines the version used on test systems.

Illustration 1 on the previous page is showing the relationships between a repository, projects, versions and channels. The project show on the left is using abstract names to explain the different parts. The project in the middle and on the right side are examples of projects which could exists in this configuration.

The following sections describe the file structure of the *repository* and the format of all required files.

File Structure

The repository structure is built in a directory which is called “repository directory” or “repository path” **A**. See [Illustration 2 on this page](#) for an overview of the repository structure.

This directory contains the `repository.xml` file **ⓑ**, which defines the projects and channels. For each project, there is a subdirectory **ⓒ** which has the same name as the identifier of the project.

In each project directory, a `info.xml` file ④ defines the project name and all versions and the order of the versions for the project. For each version, there is a sub directory ⑤, which has the same name as the version identifier.

The version directory contains a file called `files.xml` , which contains a list with all files and directories of the project version. For each file, a *SHA1 hash* is stored to compare two files for changes.

There has to be at least one package.zip file ^(H), which is an archive which contains all files of a project version. The structure in

the package file has to be exact the same as the desired installation situation. There must be no base path. A file or directory in the root path of the archive is placed inside of the installation directory.

Optional delta packages [\[J\]](#) speed up the update process from a given project version. The delta packages containing just the added or changes files from the given version and an instruction file which describes the steps to update the project from one version to another.

The sub directory file **Ⓒ** contains the file structure from the version. This directory is mainly used by the management tool to copy required files for new delta packages.

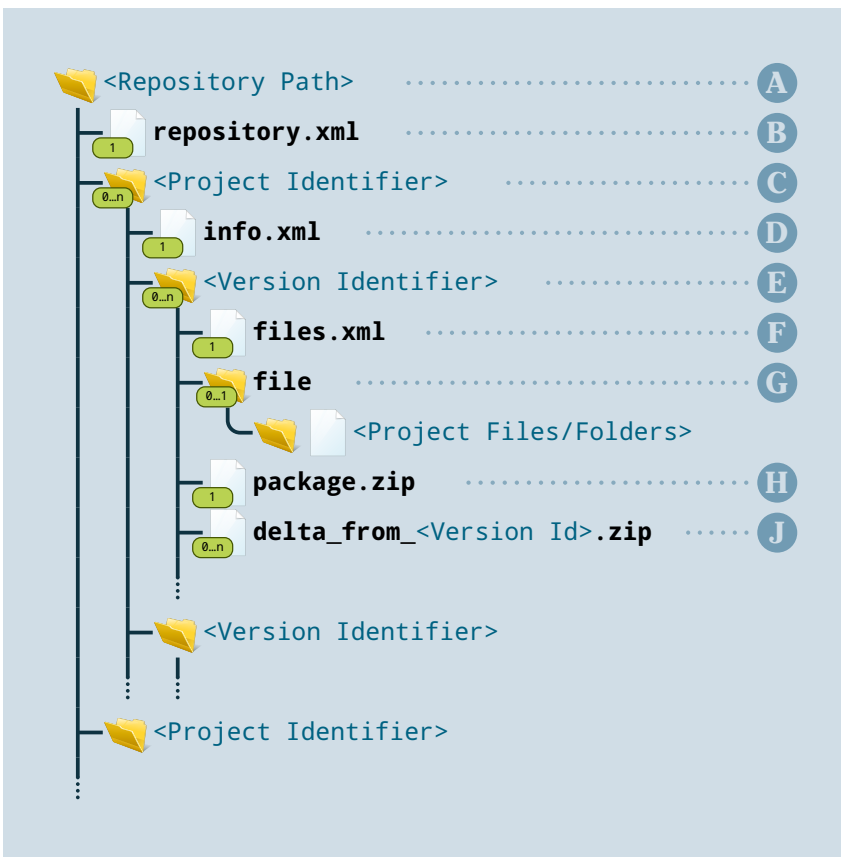


Illustration 2 The file and folder structure of the repository.

Old versions which are not changed anymore do not need this subdirectory. You can delete it to free space on the disk.

The repository.xml File

Listing 1 on this page shows the structure of the main XML file stored in the repository (see Illustration 2 on the previous page ①). The document element `Repository` has one attribute `formatVersion` which has to be set to the value 1.

This tag can contain any number of projects which are represented with the `Project` element. It has one required attribute `name`, which defines the unique *project identifier*.

The *project identifier* has to match the regular expression “[a-zA-Z][a-zA-Z0-9_]{0,63}”. That means it has to start with a letter, and contain only letters, digits and the underline character. The maximum length of the identifier is limited to 64 characters. This identifier is also used as sub directory name.

The Project Element

Each `Project` element contains the installation path, which is given in the `InstallationDirectory` element. There are two placeholders you can use to address the program directory on the client.

- `•ProgramFiles•` (• = Unicode 0x25CF)
- `•ProgramFilesX86•`

```
<?xml version="1.0" encoding="UTF-8" ?>
<Repository
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://educateit.ch/software/TopVersion/Repository/1"
  formatVersion="1">

  <Project name="...">
    <InstallationDirectory>...</InstallationDirectory>
    <Channel name="...">
      <Version>...</Version>
      <StartProgram>...</StartProgram>
      <StartArguments>...</StartArguments>
    </Channel>
    ...
  </Project>
  ...
</Repository>
```

Listing 1 Structure of the repository.xml file.

The first one is replaced by the program file directory for the respective platform. The second one is working only for a 64 bit client. There it is replaced by the 32 bit program file directory.

After the installation path, one or more Channel elements are following, which are representing a channel. After the channel element, optional Configuration elements can follow to define different configurations for the project.

The Channel Element

The Channel element has one required attribute name which defines the unique channel identifier.

It contains the Version element, which sets the current requested version for the channel. Each client which is listening to this channel will try to update to this version number.

The StartProgram element contains the absolute path to the executable which can be started after the update. You can use several placeholders in this value.

- ●ProgramFiles●
- ●ProgramFilesX86●
- ●InstallationDirectory●

The meaning of the first two is the same as described for the InstallationDirectory element. The last one is replaced by the used installation directory. This is the one you specified with the InstallationDirectory element.

Be aware that the path in the StartProgram element has to point to a *binary executable*, but not to any script which is executed by looking up some file association. To execute batch files, you have to specify at this point “cmd.exe” or for example to execute Perl files you have to point this path to the *Perl interpreter* — not to the script file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Project
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://educateit.ch/software/TopVersion/Project/1"
  formatVersion="1">

  <Name>...</Name>
  <Version>...</Version>
  <Version>...</Version>
  <Version>...</Version>
  ...
</Project>
```

Listing 2 The structure of the info.xml file.

A list of command line arguments can be passed to the executable with the `StartArguments` element.

The `info.xml` File

In each project subdirectory (see [Illustration 2 on page 11](#) ③) has to be the `info.xml` file ④, which contains information about the project human readable name and the list of all versions of this project. It also defines the order of the versions.

[Listing 2 on the previous page](#) shows the XML structure of the `info.xml` file. The `Name` element contains the human readable name of the project. This name is mainly used to display on the screen and can be different from the *project identifier*.

After the `Name` element any number of `Version` elements may follow. For a valid repository there has to be at least one version in the project. The `Version` elements contains the *version identifier* which has to match the regular expression `[-0-9a-zA-Z_.]{1,64}`. That means the *version identifier* is limited to minimum one and maximum 64 characters. It can consist of letters, digits, the minus, the underline and the point character.

The order of the version entries is critical. Older versions shall come before newer versions. *Top Version* does not interpret the version number or names in any way — it just regards the order of the *version identifiers* in this file.

The Version Sub Directory

For each version entry, there has to be a subdirectory ⑤ with the same name. This directory contains at least the `package.zip` file ⑥, which is a ZIP archive with all files from the project version.

To speed up updates, there may be any number of `delta_from_xxx.zip` files ⑦, where `xxx` stands for the version identifier of the base version from which the update is made. For example if you have two versions `a` and `b`. The version directory `b` can contain the file `delta_from_a.zip`, which contains just the difference from version `a` to version `b`.

The *Top Version manager* is also creating the subdirectory `file` ⑧, which contains the file structure of the project version, and the XML file `files.xml` ⑨ which contains a list of all files and directories of the project version with *SHA1 hashes* for each file. Because creating file hashes is slow, the manager is reading the file information from the `files.xml` file to compare versions. And it is using the files in the `file` sub directory to create the final delta package.

Delta Package Format

A delta package contains just changed and added files when updating from one version to another. These files are placed in directory structures like in the original project.

There is one special directory in the archive root called `.TopVersion` ^(K). It contains one single file `delta.xml` ^(L) with the step by step instructions for the update.

Illustration 3 on this page shows the structure inside of a delta package.

The `delta.xml` File Format

See Listing 3 on this page for the structure of the XML document. The document element of the `delta.xml` file has one required attribute `formatVersion` which has to be set to 1. It starts with two elements `FromVersion` which contains the base *version identifier* on which the delta is applied. This element is followed by

the `ToVersion` element which contains the *version identifier* of the version to which the project is updated.

After these two fixed elements, any number of *update instructions* are following. Each update instruction is a XML element which contains a parameter which is a file or directory path for all current supported elements.



Illustration 3 The structure inside of a delta package.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProjectDelta formatVersion="1">
  <FromVersion>...</FromVersion>
  <ToVersion>...</ToVersion>

  Instructions...
  ...

</ProjectDelta>
```

Listing 3 The structure of the `delta.xml` file.

The instructions are processed in the given order. For example if you delete a directory you have to delete all files from this directory first. The instructions to delete the files have to be placed before the instruction to delete the directory.

All supported instructions are shown in the following list:

- RemoveFile
- RemoveDirectory
- AddDirectory
- AddFile
- UpdateFile

Each instruction is added as a XML element with the name of the instruction. The contents of the element is the relative path to the affected file or directory with the project directory as root.

[Listing 4 on this page](#) shows an example `delta.xml` file which is using all of the instructions explained above.

If you use the instruction `AddFile` or `UpdateFile`, the delta archive has to contain the added or updated version of the file in exact the given location.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProjectDelta formatVersion="1">
  <FromVersion>1.0.0</FromVersion>
  <ToVersion>1.1.0</ToVersion>
  <RemoveFile>src/performance/performance/SystemTime.java</RemoveFile>
  <RemoveFile>src/assembly/bin.xml</RemoveFile>
  <RemoveDirectory>src/assembly</RemoveDirectory>
  <RemoveFile>KEYS</RemoveFile>
  <AddFile>newFile.txt</AddFile>
  <AddDirectory>src/folderInVersion110</AddDirectory>
  <AddFile>src/folderInVersion110/A_new_Document.txt</AddFile>
  <AddFile>src/site/newFile.txt</AddFile>
  <UpdateFile>src/ntdll/nteventlog.cpp</UpdateFile>
</ProjectDelta>
```

Listing 4 An example `delta.xml` file.

Chapter 2

Client

This chapter describes the *Top Version* client which is running on the user side and keeps the project up-to-date.

Mode of Operation

The *Top Version* update client is working with a few simple steps to check and update the project:

- Check the current installed project version.
- Get the required version from the repository.
- Compare the installed version with the required version.
- Update the project.
- Cleaning up and start the program.

The individual steps are explained in the sections below.

Check the Current Installed Version

The update client is storing the last installed version, the used channel and the parameters to start the program after an update in a XML file called *status file*. This file is located in the program data directory, which location depends on the used operating system.

For example in Windows 7, for the project Example1, this data is stored at the following location:

C:\ProgramData\EducateIT\Top Version\project\Example1.xml

Listing 5 on this page shows an example how this file can look like.

If this file is not found, it is assumed that no project version is currently installed at the installation location. In this case a full update is performed in any case.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProjectStatus
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://educateit.ch/software/TopVersion/ProjectStatus/1"
  formatVersion="1">

  <Version>1.1.0</Version>
  <Channel>Production</Channel>
  <StartProgram>E:/Test/Program/Program.exe</StartProgram>
  <StartArguments></StartArguments>
</ProjectStatus>
```

Listing 5 Example of the XML file storing the last installed version.

Get the Required Version

The repository is accessed and the `repository.xml` file is read to get the necessary information about the project and channel. If the specified project or channel is not found, the update is stopped with an error message.

With a parameter in the configuration, you can choose to start the installed program anyway, if the repository is not reachable. This is working only, if the update was done once and the program parameter were stored in the status file.

Compare Versions

Now the installed version identifier is compared with the requested version identifier. If there is no difference, no update is made and all update steps are skipped.

On a difference, the project information is read from the `info.xml` file. The order of the version identifiers is compared. If the requested identifier is newer than the installed, an update is prepared. Otherwise the update process is stopped with an error, because a downgrade has to be done manually.

It is possible to allow automatic downgrades with a parameter in the configuration.

Update Project

First, an matching delta package is searched. If one is found, a delta update is prepared. If no delta package is found, a full package is searched for a full update.

The delta or full update is applied. To deal with files which are currently in use, old files are just moved out of the way into a temporary folder on the same drive. This is usually a folder starting with `".TopVersion_"` inside of the installation directory.

After an successful update, the installed version is written in the *status file*.

Clean Up and Start Program

In this step all created temporary files and folders are removed as far this is possible. In some rare cases files which are locked by the system can not get removed automatically. This is noted in the log file.

After the cleanup, the specified program is started if the corresponding command line flag is set. This program is started as detached process, because the update application is stopped after the start of the program.

Configuration

You can see the structure of the configuration file in [Listing 6 on this page](#). The *Top Version* client is using the *Configuration B* system for the configuration. This section just describes the meaning of the configuration values, but not all features of the configuration.

Together with this manual you should get the manual “*Configuration B - Administrator Manual*”. All details about the configuration format and additional features are described there.

[Table 1 on page 23](#) is showing all possible configuration values and describes the meaning of them.

See [Listing 7 on the next page](#) and [Listing 8 on the next page](#) for example configurations.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration
  version="1"
  xmlns="http://educateit.ch/software/BlueStone/Configuration/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Module name="Application">
    <Value name="repository">...</Value>
    <Value name="dialogTitle">...</Value>
    <Value name="logFilePath">...</Value>
    <Value name="startWithErrors">...</Value>
    <Value name="allowAutomaticDowngrade">...</Value>
    <Value name="problemMessageTitle">...</Value>
    <Value name="problemMessageText"><![CDATA[
      ...
    ]]></Value>
    <Value name="problemMessageInitialWidth">...</Value>
    <Value name="problemMessageInitialHeight">...</Value>
  </Module>

</Configuration>
```

Listing 6 The structure of the client configuration file.

```

<?xml version="1.0" encoding="UTF-8" ?>
<Configuration
  version="1"
  xmlns="http://educateit.ch/software/BlueStone/Configuration/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Module name="Application">

    <Value name="repository">file:///E:/Test/TopVersionRT/Test15/repository.xml</Value>

    <Value name="dialogTitle">Update Example1</Value>
    <Value name="logFilePath">E:/Test/TopVersion/Log/TopVersion_•computerName•.log</Value>

    <Value name="startWithErrors">Yes</Value>
    <Value name="allowAutomaticDowngrade">Yes</Value>

    <Value name="problemMessageTitle">A Problem With The Update</Value>
    <Value name="problemMessageText"><![CDATA[
<h1>There was a problem with the update</h1>
<p>Please call <b>555-1234</b> to get help with an manual update of the software</p>
<p>See <a href="details">more technical details</a> about the problem.</p>
]]></Value>
    <Value name="problemMessageInitialWidth">450</Value>
    <Value name="problemMessageInitialHeight">250</Value>

  </Module>

</Configuration>

```

Listing 7 Example configuration with all values used for the *Top Version* client.

```

<?xml version="1.0" encoding="UTF-8" ?>
<Configuration
  version="1"
  xmlns="http://educateit.ch/software/BlueStone/Configuration/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Module name="Application">
    <Value name="repository">file://RepServer/TopVersionR/repository.xml</Value>
  </Module>

</Configuration>

```

Listing 8 Minimal example configuration with server path.

Value	Description
repository	This value is required and specifies the location of the repository as URL. This version is supporting only file URLs. file://server/path/repository.xml file:///C:/path/repository.xml
dialogTitle	This optional value changes the title of the update window. If you omit this parameter, the title of the dialogue is set to the name of the application.
logFilePath	This optional value sets an absolute file path to the log file. Make sure the file and directory, where this path is pointing to is writable. There is no message if the log can not be written. Make sure to check the log file for new entries if you change this value. If you omit this parameter, no log file is written.
startWithErrors	This optional flag requests a start of the program if the repository can not be read. The default value is No, which stops the client with a problem message if the repository is not readable. To enable this feature, you have to set this value to Yes. If the repository is reachable, but there are problems with the update which could leave the project in a undefined state, the process is stopped with a problem message in any case.
allowAutomaticDowngrade	This optional flag enables automatic downgrades. Normally if you request an update from a newer to an older version, the client will stop with a problem report. If you set this flag to Yes, the downgrade will be done automatically.
problemMessageTitle	This value is optional. If you configure this value, the title of an problem message is replaces with this text. The text is limited to 128 characters.
problemMessageText	This value is optional. If you configure this value, the message of the problem message is replaced with this text. You can use <i>HTML</i> codes to format the text. The special link <code>...</code> displays the technical details of the problem message. If you omit this link, there is no way to display the details of the error message.
problemMessageInitialWidth problemMessageInitialHeight	This value is optional. The problem message dialogue does not change the size automatically to your context. Set here a new value for the initial width or height. The value has to be in the range between 200 and 1024.

Table 1 The configuration values for the client.

Command Line Arguments

There are several command line arguments you can specify. See [Table 2 on this page](#) for a complete list of all arguments with a description.

The two most important and required arguments are `--project` and `--channel` which are selecting the project and channel for the update.

[Listing 9 on this page](#) shows some example how to start the update client.

```
TopVersion.exe --project=Example1 --channel=Production
TopVersion.exe -p=Example1 -c=Production --startAfterUpdate
TopVersion.exe -p=Example2 -c=Beta --forceFull
TopVersion.exe -p=Example2 -c=Beta -f
```

Listing 9 Examples how to start the update client.

Argument	Description
<code>--project</code> <code>-p</code>	With this required parameter you specify the project identifier of the project you would like to update.
<code>--channel</code> <code>-c</code>	With this required parameter you select the channel for the update.
<code>--startAfterUpdate</code> <code>-s</code>	If you set this flag, the program you configured for the selected project is automatically started after a successful update or if no update is necessary. In case of any problem, an message with the problem is shown to the user but the program is not started.
<code>--forceFull</code> <code>-f</code>	If you add this flag, the project is updated in every case to the version which is specified in the selected channel. Always a full update is done, removing the previous contents from the installation location. Use this to repair a corrupt installation, to force a downgrade if you like to switch from one channel to another.
<code>--keepUiOpen</code>	This flag exists mainly for debugging purposes. If you add this flag, the user interface is kept open after the update and you can story the progress details.

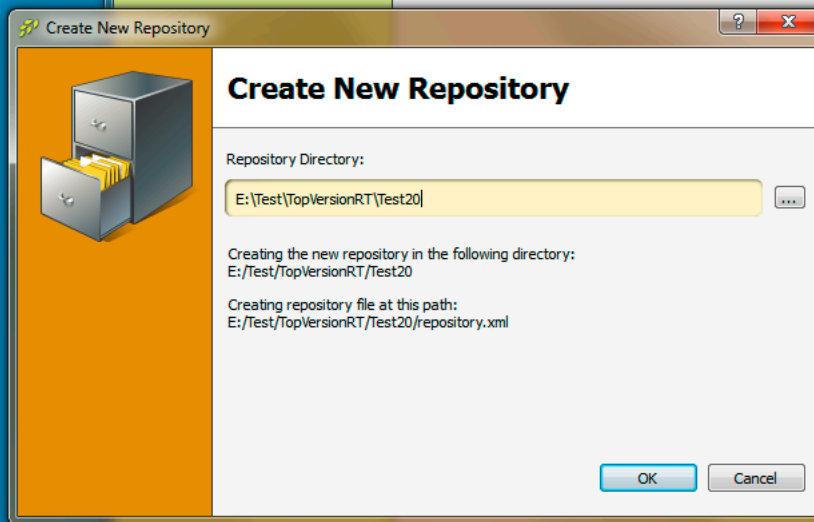
Table 2 Command line parameters for the client.

Chapter 3 **Manager**

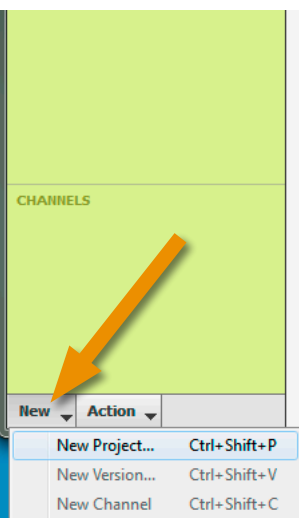
This chapter describes the *Top Version* manager.



Screen 1: The welcome page.



Screen 2: Create a new repository dialogue.



Screen 3: Use the action drop down to create a new project.

Creating a New Repository

After the first start of the repository manager you see the welcome screen (Screen 1). Here you have the choice to open an existing repository or to create a new one.

Click on the button labelled “Create New Repository...” to create a new repository. A dialogue appears (Screen 2) where you have to choose the path to the new repository. This has to be an empty folder somewhere.

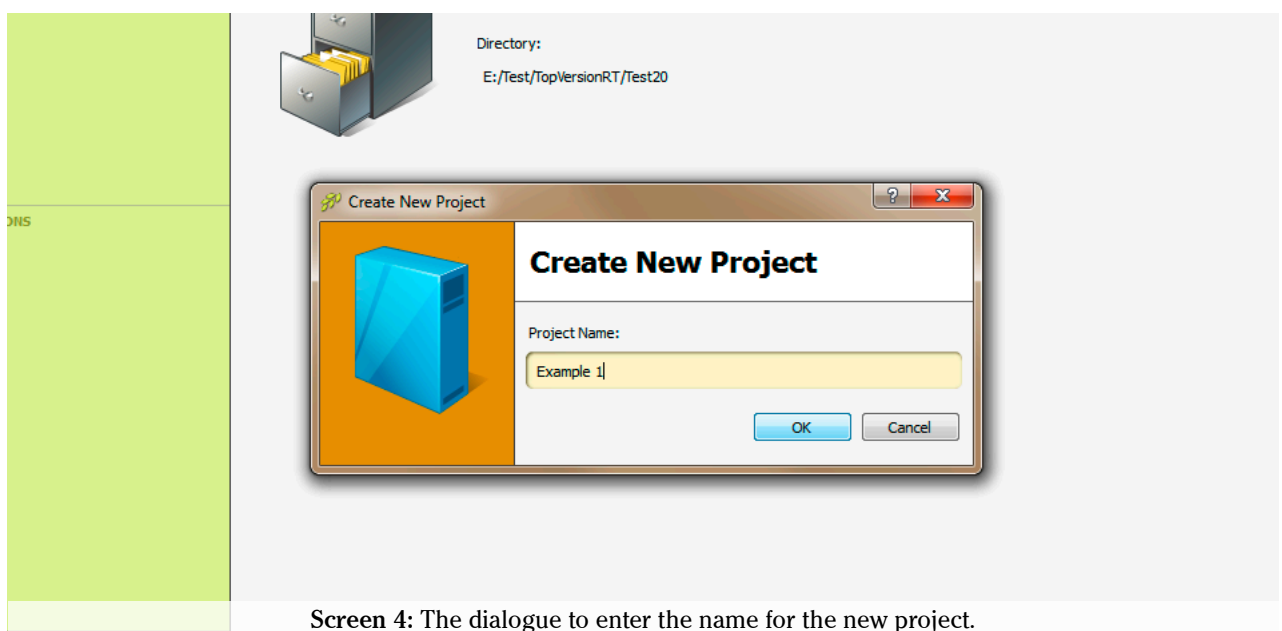
A click on the small button on the right side of the field opens a directory selection dialogue where you can choose and create a new folder. In case of an error, it is written in red below the field. If the path is usable, you see a preview where the repository XML file will be created.

Create a New Project

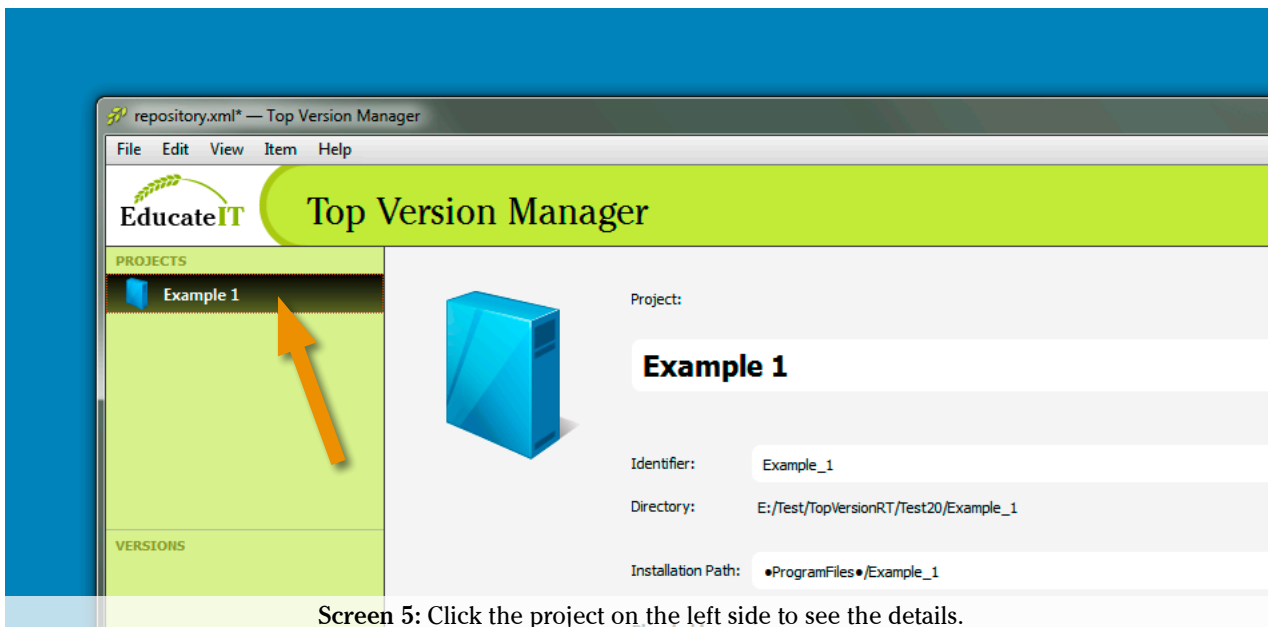
After creating the repository you see the blank repository screen. To create a new project, choose “New Project...” from the “New” drop down in the bottom left corner of the window (Screen 3).

A new dialogue appears (Screen 4) in which you have to enter the name of the new project. You enter the human readable name in this field. The repository manager will automatically create an new identifier from it.

Clicking on “OK” will create the new project.



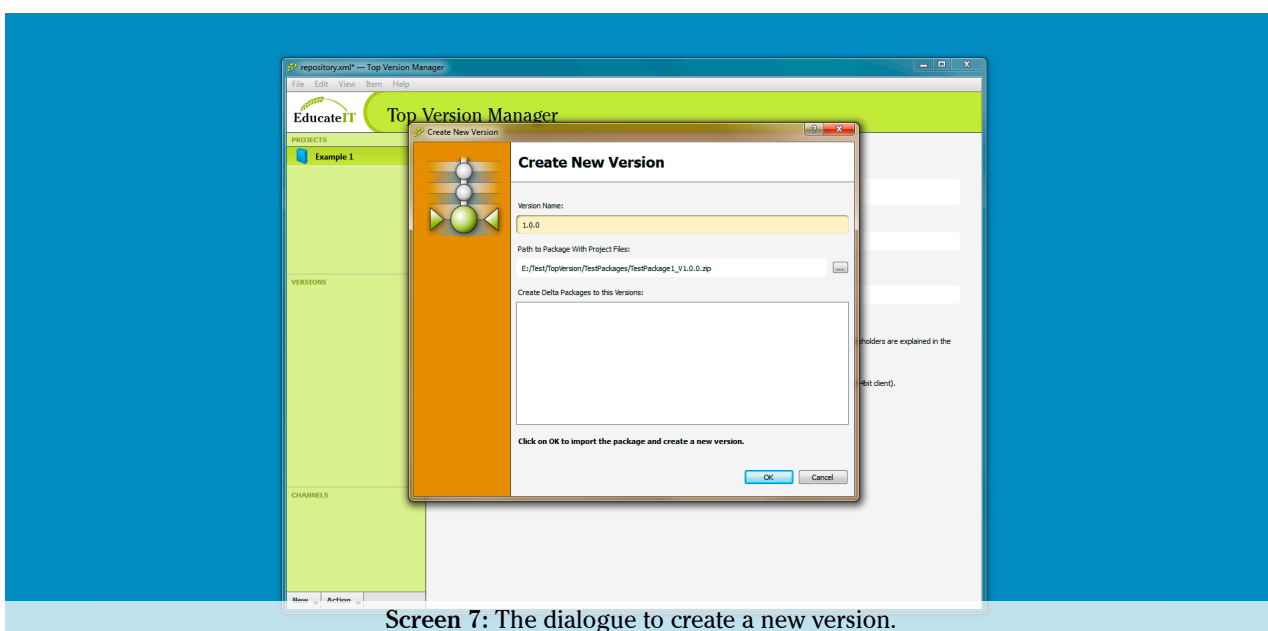
Screen 4: The dialogue to enter the name for the new project.



Screen 5: Click the project on the left side to see the details.



Screen 6: While the project is not saved, you can edit the project identifier.



Screen 7: The dialogue to create a new version.

The new project is shown in the project list on the left side of the window (Screen 5). If you click it you can see all details of the project in the main area of the window.

As long the changes are not saved, you can edit the identifier of the project (Screen 6). Once you save the project, the identifier can not be edited anymore.

Create a New Version

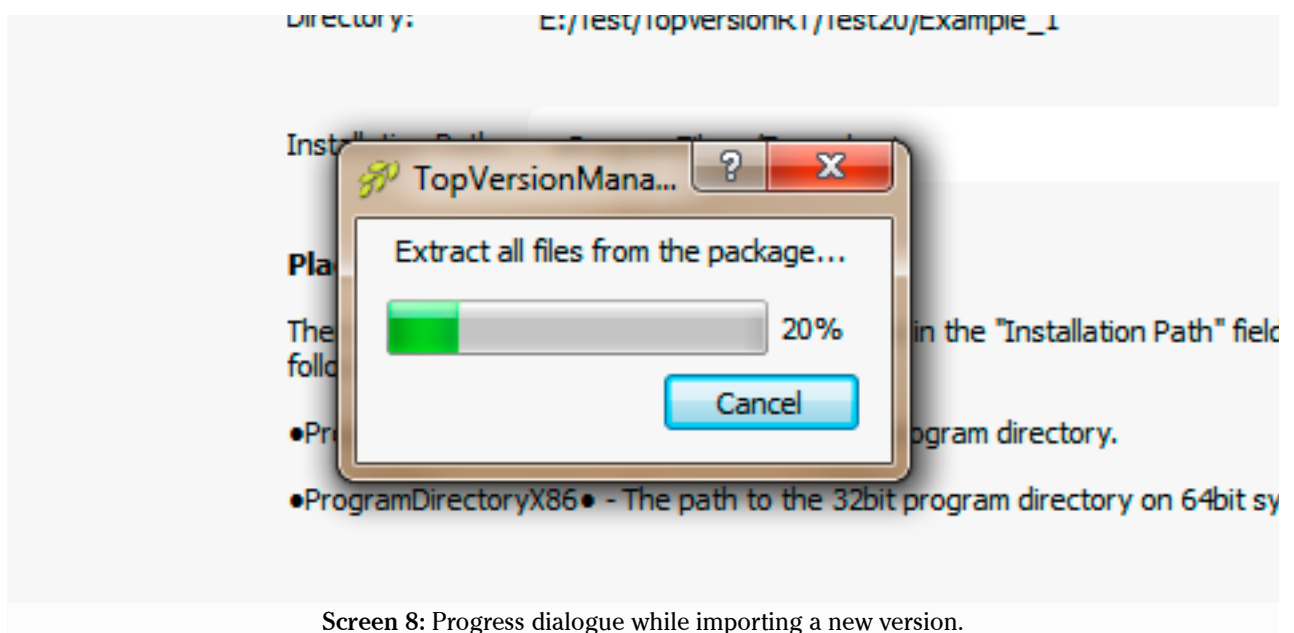
To create a new version you can use the “New” drop down in the bottom left corner of the window. Select “New Version...” in the pop-up menu. All commands are also available in the application menu.

A new dialogue opens (Screen 7) where you have to enter the version identifier and select the path to the archive which contains the files for the version.

If you have previous versions, you can select in the list below, for which versions you would like to create a new delta package. By default the last 5 versions are selected, but you can select any combination of versions.

When you click on the “OK” button, the version data is imported and the delta packages are created (Screen 8). This can take a while, especially when you work on a network share.

If any problem is detected while you import a new version, and a problem message is shown. The repository is not changed in this case and you can fix the problem and try again.



Screen 8: Progress dialogue while importing a new version.

Create a Channel

After you added at least one version to your project, you can add a channel to the project.

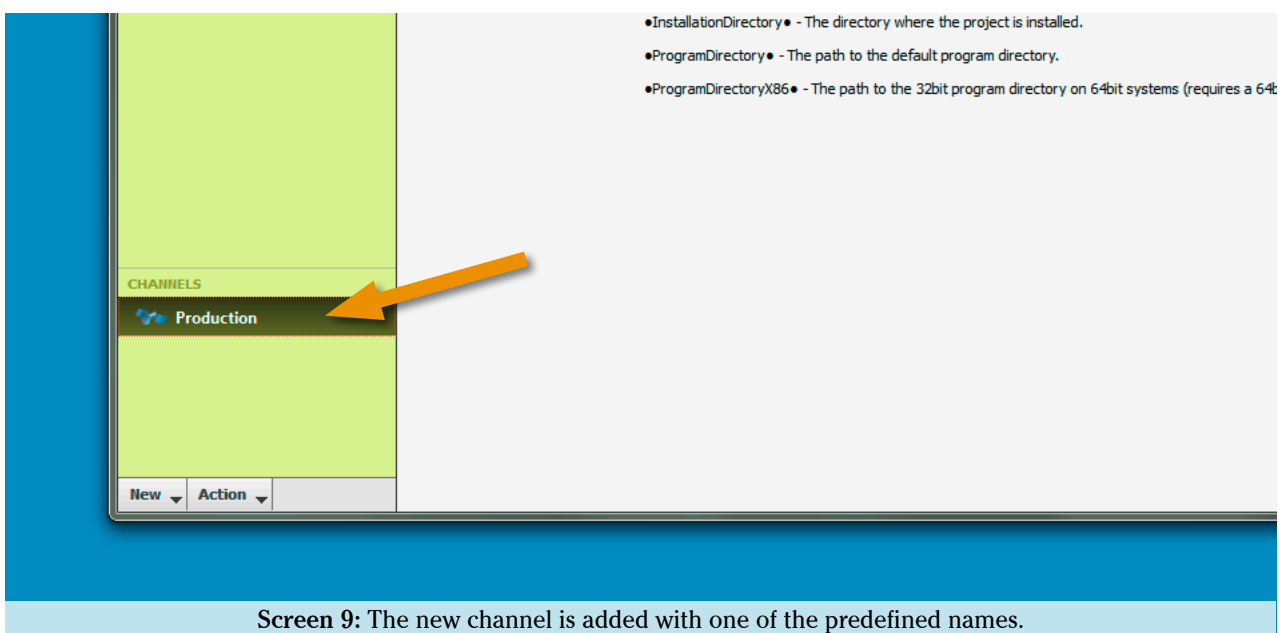
Use the “New” drop down in the bottom left corner and select “New Channel...”. A new channel is added, using one of the pre-defined names (Screen 9).

You can change this name on the channel page. Do this by clicking on the channel in the list on the left side of the window.

Save the Repository

If you select “File”, “Save” all changes you made to the repository are made permanent. Until you save, you can undo every step and rename some identifiers. This is not possible after saving your changes.

In the *very rare* case of a problem while saving the version, a problem dialogue is shown. Make sure that you look at the details of the problem in this dialogue. One message is telling you if the structure of the repository could get restored or not. If not, please restore the previous version from a backup before you continue.



Features and Preferences

This section describes some important features in the user interface of the manager, which are not self explanatory.

Warn Sign on Changed Version

If you change the version in a channel, a warn sign appears at the right side of the channel name. This sign shall remember you that when you save your changes, probably all clients which are configured for this channel will update to the new version.

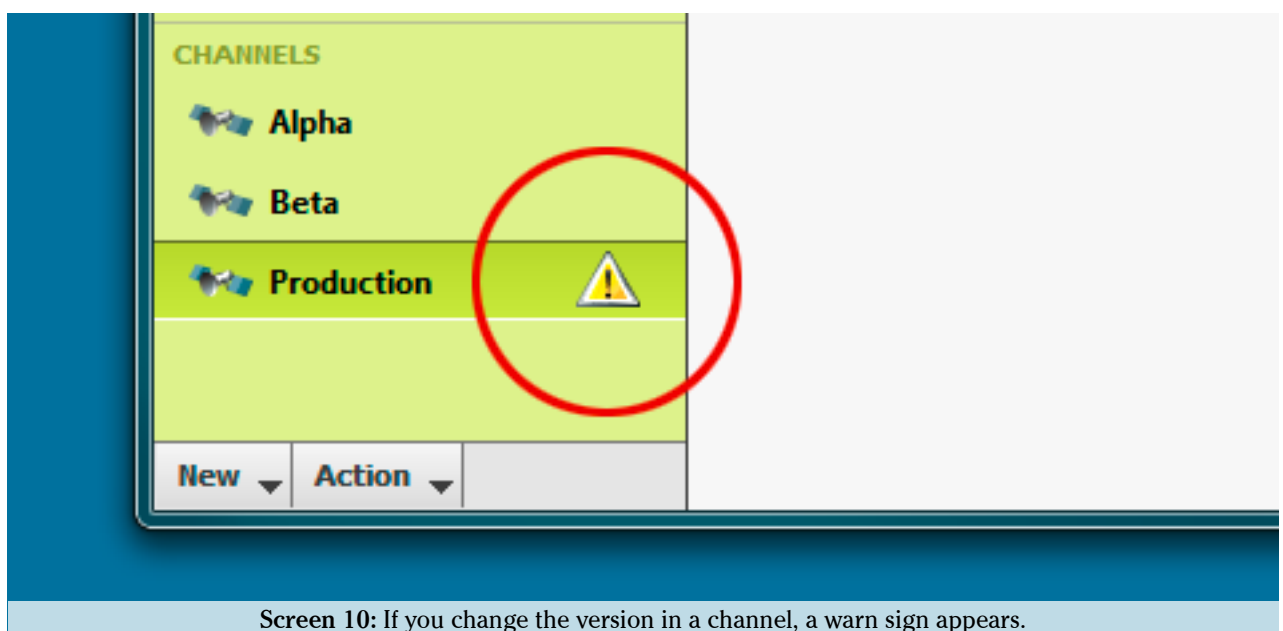
See Screen 10 how the warn sign is displayed.

The warn sign is also displayed if you add a new channel to the repository, until you save your changes.

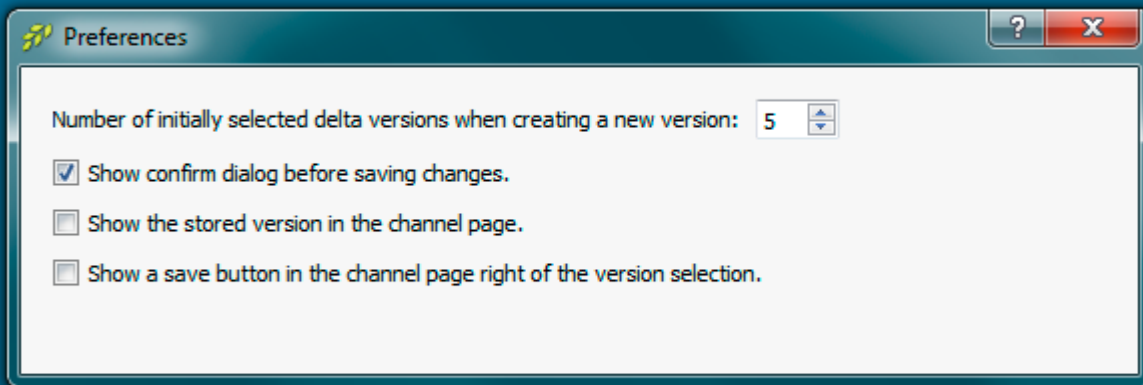
Preferences

There are some features which can only be enabled or disabled in the preferences of the user interface. See Screen 11 for the preferences window.

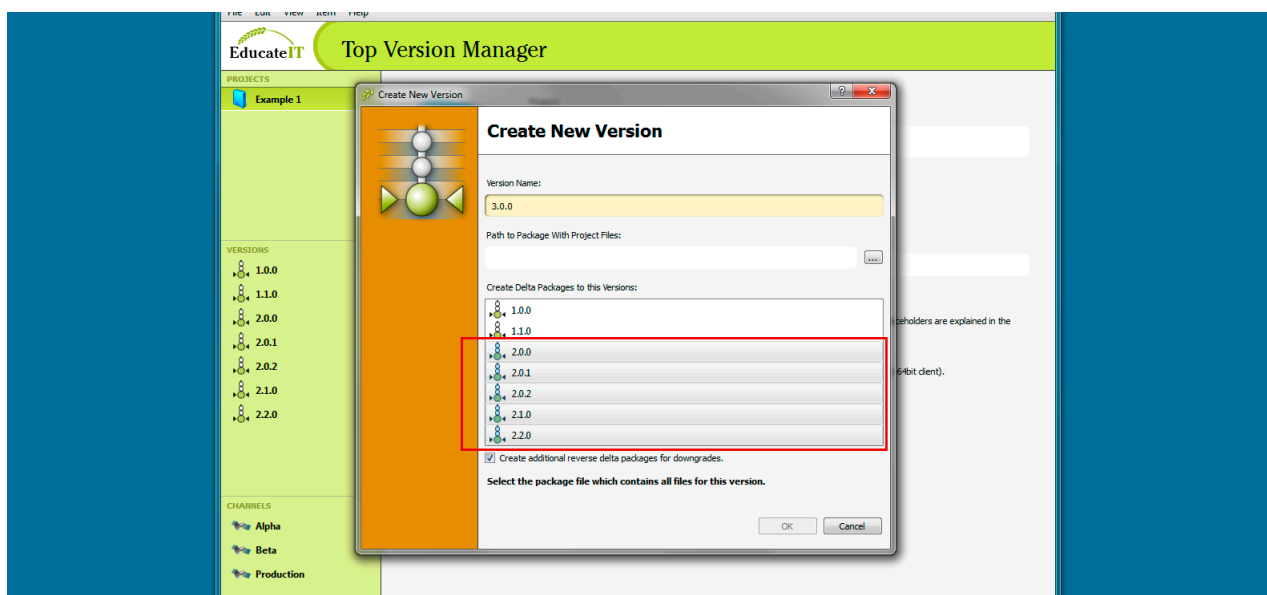
The entries in this preferences window are described in the following sections.



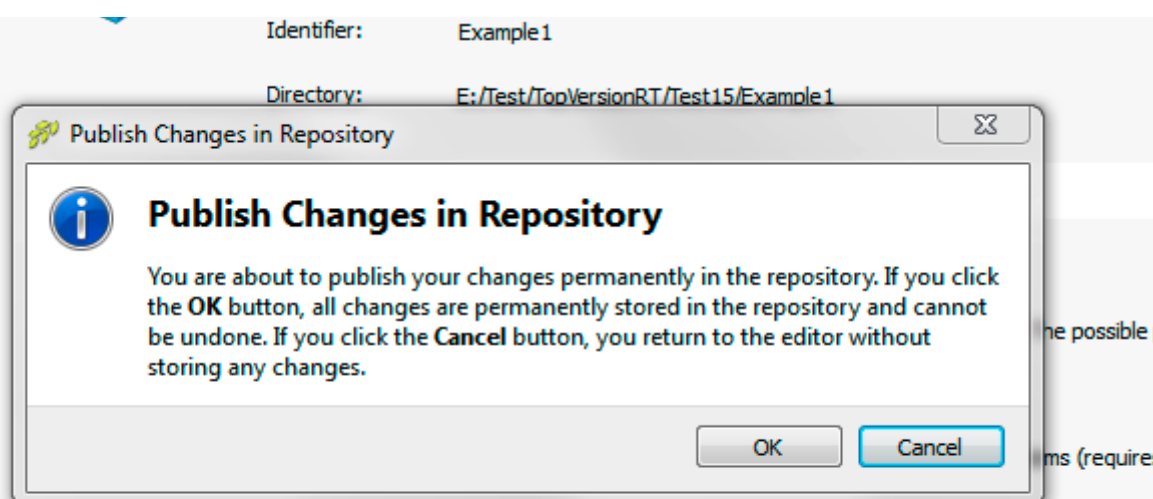
Screen 10: If you change the version in a channel, a warn sign appears.



Screen 11: The window with the preferences for the user interface.



Screen 12: You can change the number of automatically selected delta versions in the preferences window.



Screen 13: The confirmation dialogue shown before you save your changes into the repository.

Number of Initially Selected Delta Versions

The first entry in the preferences window changes the number of automatically selected newest versions to create delta packages from. See Screen 12 with an example of five selected delta versions.

If you change the number to zero, no packages are selected by default when you create a new version.

Show Confirm Dialogue Before Saving Changes

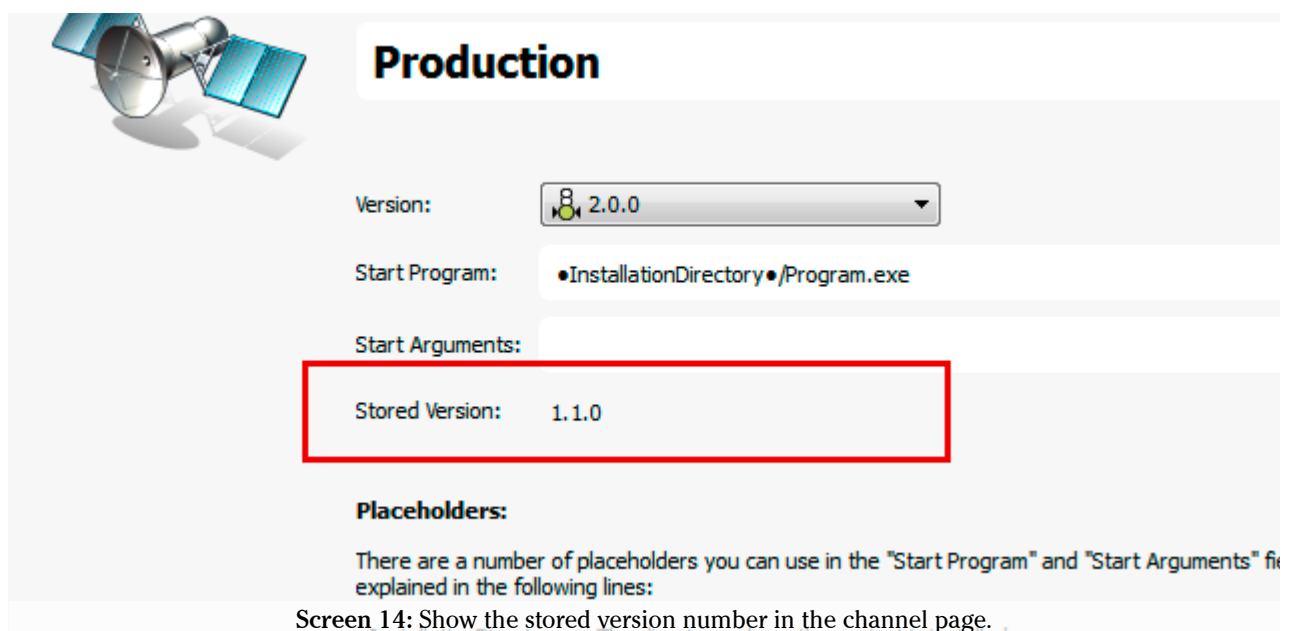
When this check box is checked, a confirmation dialogue is shown when you save your changes. See Screen 13 for the confirmation dialogue. You have to confirm, that you like to save all made changes into the repository. At this point you have the choice to cancel, and leave the repository untouched.

Show the Stored Version in the Channel Page

If you check this check box, the currently stored version number is shown in the channel page. See Screen 14 for an example.

Show a save button in the Channel Page...

If you check this check box, a save button is displayed in the channel page, just right of the version selection. This save button has the same functionality as if you select Save from the menu, or are pressing Ctrl+S.

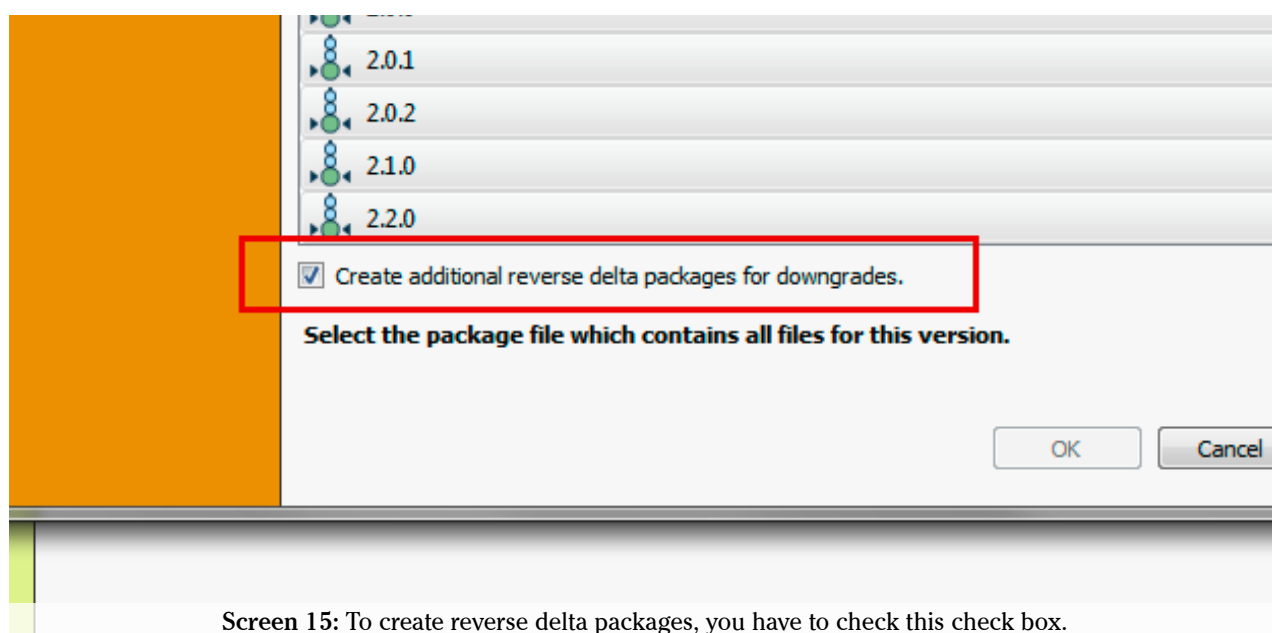


Creating Reverse Deltas

If you allow automatic downgrades to older versions, you also should create reverse delta packages to speed up the downgrade to previous versions.

Reverse delta packages are like regular delta packages, but they are placed in the directories of the previous versions. They provide all necessary informations to make an update from the new added version back to the selected version.

To automatically create a forward and a reverse delta package for each selected version, you have to check the check box below the list of versions in the new version window. See Screen 15 where this check box is checked.



Screen 15: To create reverse delta packages, you have to check this check box.

Chapter 4

Appendix

This appendix contains the XML schema for the different XML files in the repository. You should get this schema together with the software to verify the XML documents if you make manual changes or using own software to modify the repository.

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/Repository/1"
  xmlns="http://educateit.ch/software/TopVersion/Repository/1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="VersionName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[-0-9a-zA-Z_.]{1,64}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="EntryName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z0-9_]{0,63}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Channel">
    <xs:sequence>
      <xs:element name="Version" type="VersionName" />
      <xs:element name="StartProgram" type="xs:string" />
      <xs:element name="StartArguments" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="name" type="EntryName" use="required" />
  </xs:complexType>

  <xs:complexType name="Project">
    <xs:sequence>
      <xs:element name="InstallationDirectory" type="xs:string" />
      <xs:element name="Channel" type="Channel" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="EntryName" use="required" />
  </xs:complexType>

  <xs:complexType name="Repository">
    <xs:sequence>
      <xs:element name="Project" type="Project" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="formatVersion" type="xs:integer" use="required" />
  </xs:complexType>

  <xs:element name="Repository" type="Repository" />
</xs:schema>

```

Listing 10 Schema for the repository.xml file.

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/Project/1"
  xmlns="http://educateit.ch/software/TopVersion/Project/1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="VersionName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[-0-9a-zA-Z_.]{1,64}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Project">
    <xs:sequence>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Version" type="VersionName" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="formatVersion" type="xs:integer" use="required" />
  </xs:complexType>

  <xs:element name="Project" type="Project" />
</xs:schema>

```

Listing 11 Schema for the project info file info.xml.

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/ProjectVersion/1"
  xmlns="http://educateit.ch/software/TopVersion/ProjectVersion/1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="VersionName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[-0-9a-zA-Z_.]{1,64}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="HashType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="SHA1" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Hash">
    <xs:simpleContent>
      <xs:extension base="xs:string" >
        <xs:attribute name="type" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="Directory">
    <xs:sequence>
      <xs:element name="Path" type="xs:string" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="File">
    <xs:sequence>
      <xs:element name="Path" type="xs:string" />
      <xs:element name="Size" type="xs:integer" />
      <xs:element name="Hash" type="Hash" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ProjectVersion">
    <xs:sequence>
      <xs:element name="Version" type="VersionName" minOccurs="0" maxOccurs="unbounded" />
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="File" type="File" />
        <xs:element name="Directory" type="Directory" />
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="formatVersion" type="xs:integer" use="required" />
  </xs:complexType>

  <xs:element name="ProjectVersion" type="ProjectVersion" />
</xs:schema>

```

Listing 12 Schema for files.xml file.

```

<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/ProjectStatus/1"
  xmlns="http://educateit.ch/software/TopVersion/ProjectStatus/1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="VersionName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[-0-9a-zA-Z_.]{1,64}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="EntryName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z0-9_]{0,128}" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="ProjectStatus">
    <xs:sequence>
      <xs:element name="Version" type="VersionName" />
      <xs:element name="Channel" type="EntryName" />
      <xs:element name="StartProgram" type="xs:string" minOccurs="0" />
      <xs:element name="StartArguments" type="xs:string" minOccurs="0" />
    </xs:sequence>
    <xs:attribute name="formatVersion" type="xs:integer" use="required" />
  </xs:complexType>

  <xs:element name="ProjectStatus" type="ProjectStatus" />
</xs:schema>

```

Listing 13 Schema for the installation status file.



EducateIT GmbH

Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland

Phone: +41 32 355 39 00

E-Mail: info@educateit.ch

Web: <http://www.educateit.ch/>

Author: Tobias Erbsland
Layout & Design: Tobias Erbsland
Last Modification: 2012-02-22 16:31
File Name: Top Version - Administrator Manual
Copyright: © 2011 by EducateIT GmbH