Top Version

Administrator Manual

Manager 2.0 • Client 2.0
Thin Client 2.0 • Client Service 2.0
Selector Service 1.0 • Selector Client 1.0

EducateIT

EducateIT GmbH
Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland
Phone: +41 32 355 39 00
E-Mail: info@educateit.ch
Web: http://www.educateit.ch/

Top Version – Administrator Manual
Copyright: ©2016 by EducateIT GmbH

EducateIT GmbH
Steinenbergstrasse 7
CH-4532 Feldbrunnen
Switzerland

Phone: +41 32 355 39 00
E-Mail: info@educateit.ch
Web: http://www.educateit.ch/

Illustrations and Text: TE
Layout and Design: TE
Last Modification: 2016-11-04 17:27

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by and means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the EducateIT GmbH. The EducateIT GmbH assumes no responsibility for errors or omissions, or for damages that may result from the use of information (especially including example configurations and listings) contained herein.

Contents

Preface 7
Features 7
Version for this Manual 8
See Also 8

Repository **9**

Introduction 11

File Structure 13
The `repository.xml` File 15
The `info.xml` File 16
The Version Sub Directory 16

Delta Package Format 19
The `delta.xml` File Format 19

Client **23**

Mode of Operation 25
Check the Current Installed Version 25
Get the Required Version 26
Compare Versions 26
Update Project 26
Clean Up and Start Program 26

Configuration 27
`repository` 27
`dialogTitle` 27
`startWithErrors` 27
`allowAutomaticDowngrade` 29

problemMessageTitle 29
problemMessageText 29
problemMessageInitialWidth 30
problemMessageInitialHeight 30

Command Line Arguments 31

Thin Client 33

Preface 35

Mode of Operation 37
Start-Up 37
Application Start 37
Monitoring the Process 37
Notifications 38
Application Termination 39

Command Line Arguments 41

Client Service 43

Preface 45

Configuration 47
ServerInterface 47
ServerPort 47
ApplicationList 47
ApplicationList » Identifier 50
ApplicationList » Repository 50
ApplicationList » Project 50
ApplicationList » Channel 51
ApplicationList » StartWithErrors 51
ApplicationList » AllowAutomaticDowngrade 51
ApplicationList » UpdateCheck 51

Update: Mode of Operation 53

Manager 57

Creating a New Repository 59
- Create a New Project 59
- Create a New Version 61
- Create a Channel 62
- Save the Repository 62

Features and Preferences 63
- Warn Sign on Changed Version 63
- Preferences 63

Creating Reverse Deltas 67

Add a Repository Name 69

Selector System 71

Mode of Operation 73
- System Elements 73
- Information Flow 73

Client Usage 75
- The State of the Change Button 75
- Clicking the Change Button 76
- After Confirming a Change 76

Security and Protocol 77
- Protocol Overview 77
- Authentication 77
- Risks and Weaknesses 78
- Best Practices 79

Client Configuration 81

host 81
port 81
apiKey 81
apiSecret 81
profile 81

Client Command Line 83

Service Configuration 85
The network List 85
The secrets List 87
The profiles List 88
Statistic Server Support 89
Example Configuration 89

The Profile Definition 91
The Profile Element 91
The Project Element Structure 93
The Switches 95
Profile Verification 96
Perfomed Checks 96

Appendix **97**

Preface

The *Top Version* system simplifies frequent updates of a software on a system. A software package, called “project” in this system, can have multiple versions and distribution channels. Speed and robustness were the goal while designing this system.

There is a simple 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.

For fully automatic updates, or updates on multiuser systems, there is a special combination of a thin client with a client service which is running in the background on a system.

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 thin client, the client service and the usage of the repository manager. It describes also the structure and format of the repository.

Features

- Fast and robust software or file updates for projects.
- Delta update which only updates 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.
- Simple client software to bundle with a project.
- Advanced thin client and service solution for fully automatic updates in the background.
- Selector system to provide a restricted interface for version changes. It also allows users with lesser file system rights to change a version in a safe and controlled way.

Version for this Manual

The version of each described product is shown on the title page. Make sure you use the show versions of these products, or newer ones.

See Also

All parts of this software are 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 along with this manual. If not, please request it from us.

The client service also uses the *EducateIT Service Framework* and the *EducateIT Log System*. Both systems provide a number of features which are fully explained in their own manuals. You should get these documents along with this manual. If not, please request them from us.

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 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 exists in parallel.

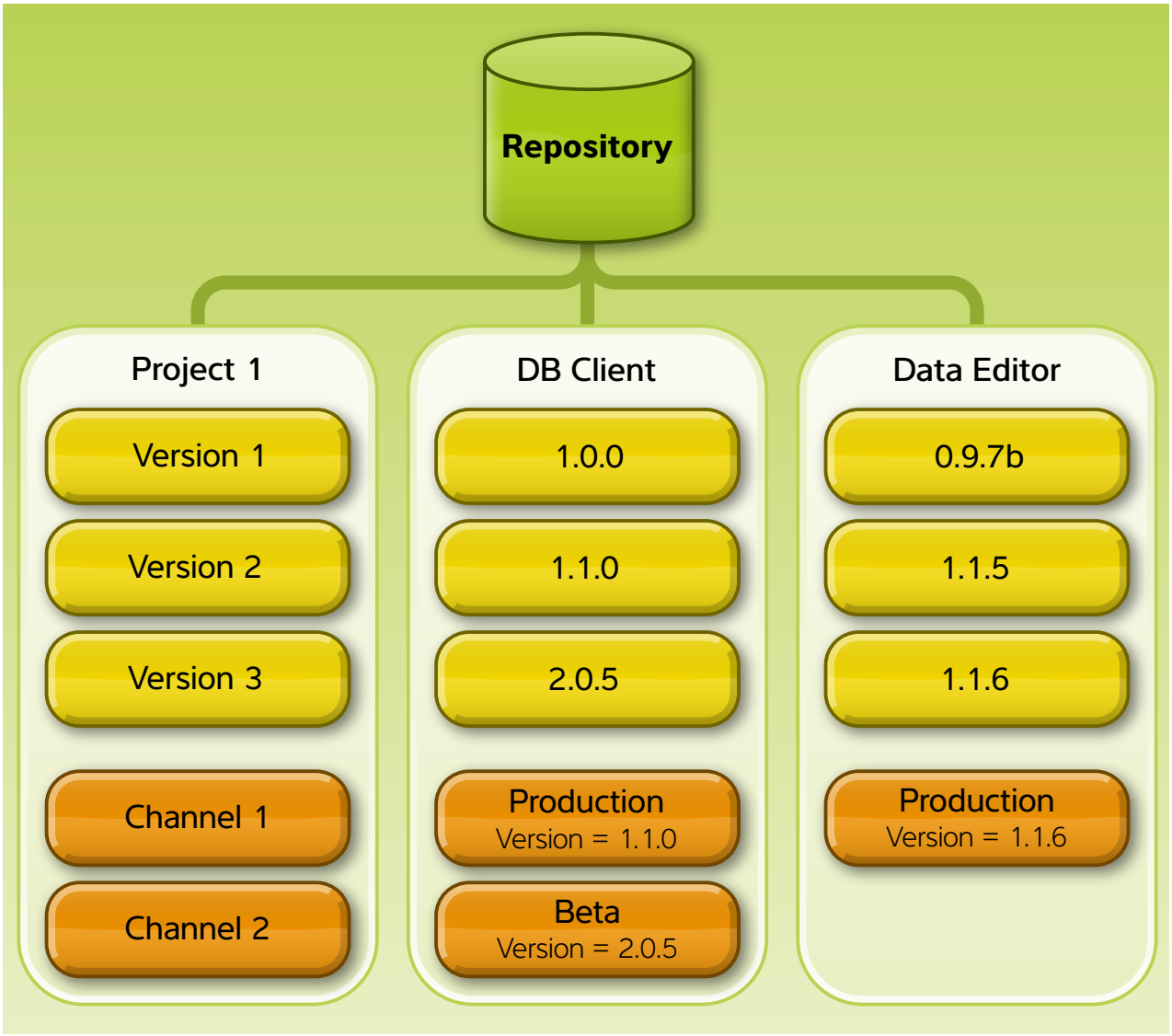


Illustration 1 The used terms and how they are related to the repository.

Each version has a unique name and a given position in the *version vector*.

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 has 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 page 11 is showing the relationships between a repository, projects, versions and channels. The project shown 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” Ⓐ. See Illustration 2 on page 13 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 sub directory Ⓒ which has the same name as the identifier of the project.

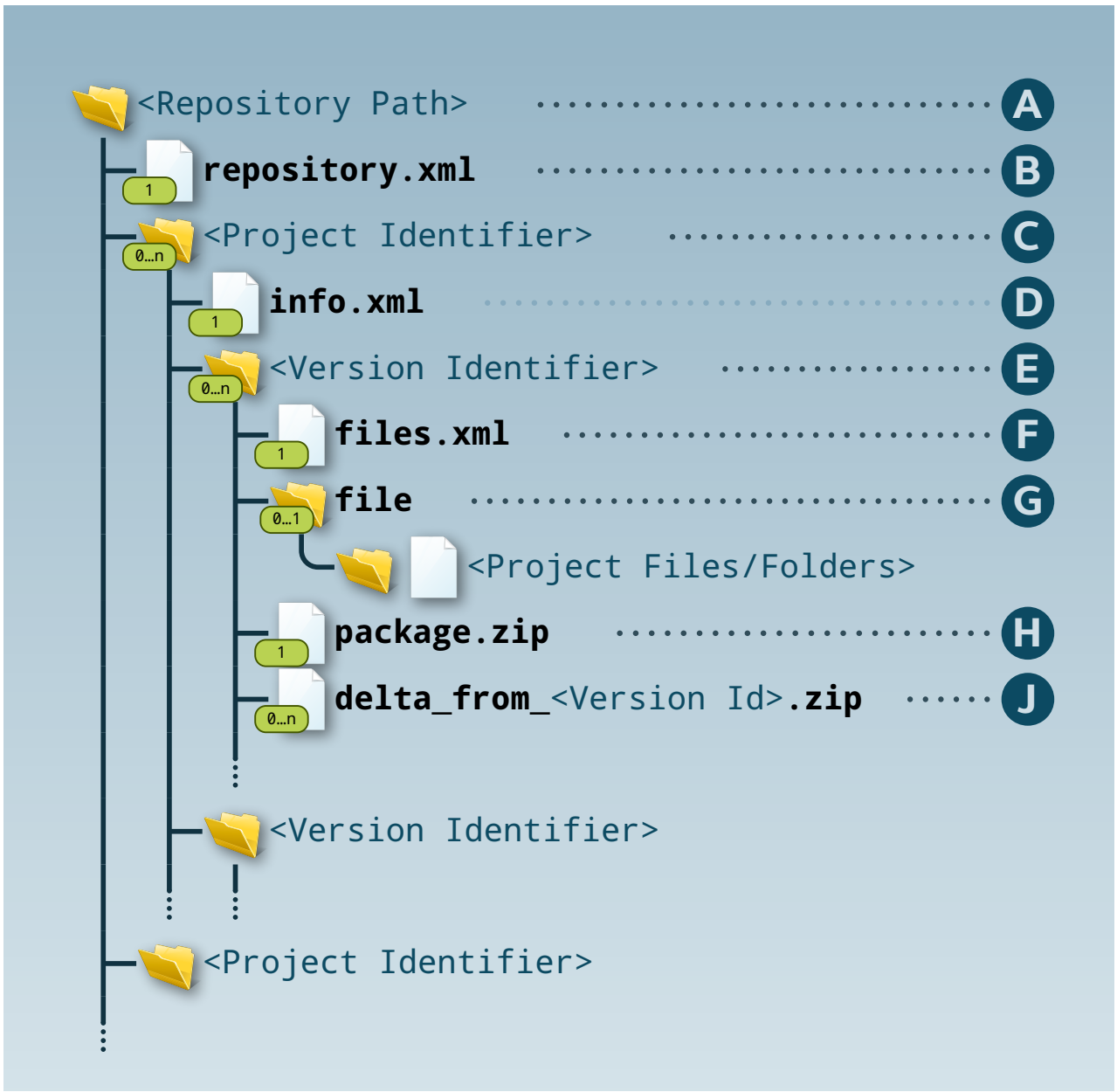


Illustration 2 The file and folder structure of the repository.

In each project directory, a `info.xml` file Ⓓ defines the project name and all versions and the order of the versions for the project. For each version, there is a sub directory Ⓔ, 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 Ⓗ, 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 Ⓙ 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.

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

```xml
<?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 `repository.xml` File

Listing 1 on page 14 shows the structure of the main XML file stored in the repository (see Illustration 2 on page 13 Ⓐ). 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. Id 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 `InstallatonDirectory` element. There are two placeholders you can use to address the program directory on the client.

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

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.

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 13 Ⓒ) has to be the `info.xml` file Ⓓ, 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 page 17 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 Ⓔ with the same name. This directory contains at least the `package.zip` file Ⓗ, 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 Ⓙ, 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`.

```xml
<?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.

The *Top Version manager* is also creating the subdirectory `file` Ⓖ, which contains the file structure of the project version, and the XML file `files.xml` Ⓕ 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. This files are placed in directory structures like in the original project.

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

Illustration 3 on page 19 shows the structure inside of a delta package.

The `delta.xml` File Format

See Listing 3 on page 20 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 this 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 page 21 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
<?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 page 25 shows an example how this file can look like.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<ProjectStatus
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://educateit.ch/software/TopVersion/ProjectStatus/2"
  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.

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.

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 "`.TopVerson_`" 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

The structure of the configuration file shown Illustration 4 on page 28. The *Top Version* client is using the *Configuration B* system for the configuration.

You should got the manual "*Configuration B - Administrator Manual*" along with this document. All details about the configuration format and additional features are described there.

See Listing 6 on page 29 for a complete example configuration which defines every single value. This is not a very likely configuration, usually you define just the values which are really necessary for your usage. Therefore with Listing 7 on page 30 minimal configuration example is provided. This minimal example defines just the repository, which is the only required value of the configuration.

The meaning of the individual values of the configuration are described in the following sections.

repository

This value is required and specifies the location of the repository as an absolute path to the XML file. Here are two examples of valid repository paths:

```
//server/path/repository.xml
```

```
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.

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 in any case with a problem message.

Module	Application
Value	`allowAutomaticDowngrade` Optional Boolean
Default	No
Value	`dialogTitle` Optional String
Constraints	▪ Maximum length: 128 character(s).
Value	`problemMessageInitialHeight` Optional Integer
Constraints	▪ Minimum value: 200 ▪ Maximum value: 1024
Default	250
Value	`problemMessageInitialWidth` Optional Integer
Constraints	▪ Minimum value: 200 ▪ Maximum value: 1024
Default	450
Value	`problemMessageText` Optional String

Default	`<h2>Problem With Update</h2>`
	`<p>There was a problem with the program update. The installation may be in a undefined state.</p>`
	`<p>See <a href="details">more technical details</a> about the problem.</p>`

Value	`problemMessageTitle` Optional String
Constraints	▪ Maximum length: 128 character(s).
Default	Problem With Update
Value	`repository` String
Constraints	▪ Must not be empty.
Value	`startWithErrors` Optional Boolean
Default	No

Illustration 4 The visualization of the `Application` configuration module of the client.

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

If you configure this value, the title of an problem message is replaced with this text. The text is limited to 128 characters.

problemMessageText

If you configure this value, the message of the problem message is replaced with this text. You can use *HTML* codes to format the text. The special link `<a href="details">...</a>` 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.

Make sure the escape the greater and lesser sign of HTML tags in the configuration or enclose the whole text into `<![CDATA[` and `]]>` brackets as shown in the example.

```
<?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">//server01/TopVersionRepo/ProjectA/repository.xml</Value>
    <Value name="dialogTitle">Update Example1</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 6 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">//server01/TopVersionRepo/ProjectA/repository.xml</Value>
  </Module>

</Configuration>
```

Listing 7 Minimal example configuration with server path.

problemMessageInitialWidth

The problem message dialogue does not change the size automatically to your content. Set here a new value for the initial width. The value has to be in the range between 200 and 1024.

problemMessageInitialHeight

The problem message dialogue does not change the size automatically to your content. Set here a new value for the initial height. The value has to be in the range between 200 and 1024.

Command Line Arguments

There are several command line arguments you can specify. See Table 1 on page 31 for a complete list of all arguments with a detailed description.

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

Listing 8 on page 31 shows a list of command line examples how to start the update client with different parameters.

```
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 8 Examples how to start the update client.

Argument	Description
`--project` `-p`	With this required parameter you specify the project identifier of the project you would like to update.
`--channel` `-c`	With this required parameter you select the channel for the update.
`--startAfterUpdate` `-s`	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, a message with the problem is shown to the user but the program is not started.
`--forceFull` `-f`	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.
`--keepUiOpen`	This flag exists mainly for debugging purposes. If you add this flag, the user interface is kept open after the update and you can study the progress details.

Table 1 Command line parameters for the client.

Chapter 3: Thin Client

The thin client is an alternative to the standalone client. This chapter describes the thin client in detail.

Preface

The “thin client” is an alternative to the standalone client. It requires an installed version of the “client service” which is described in the chapter “Client Service” on page 43.

There are a number of features which are only provided by the thin client with the client service combination:

- **Privileges separation:** The thin client needs no additional rights and is running in the context of the user of the application. It especially need no write rights to the application directory. Only the service which is running on the system needs write access to the application directory.

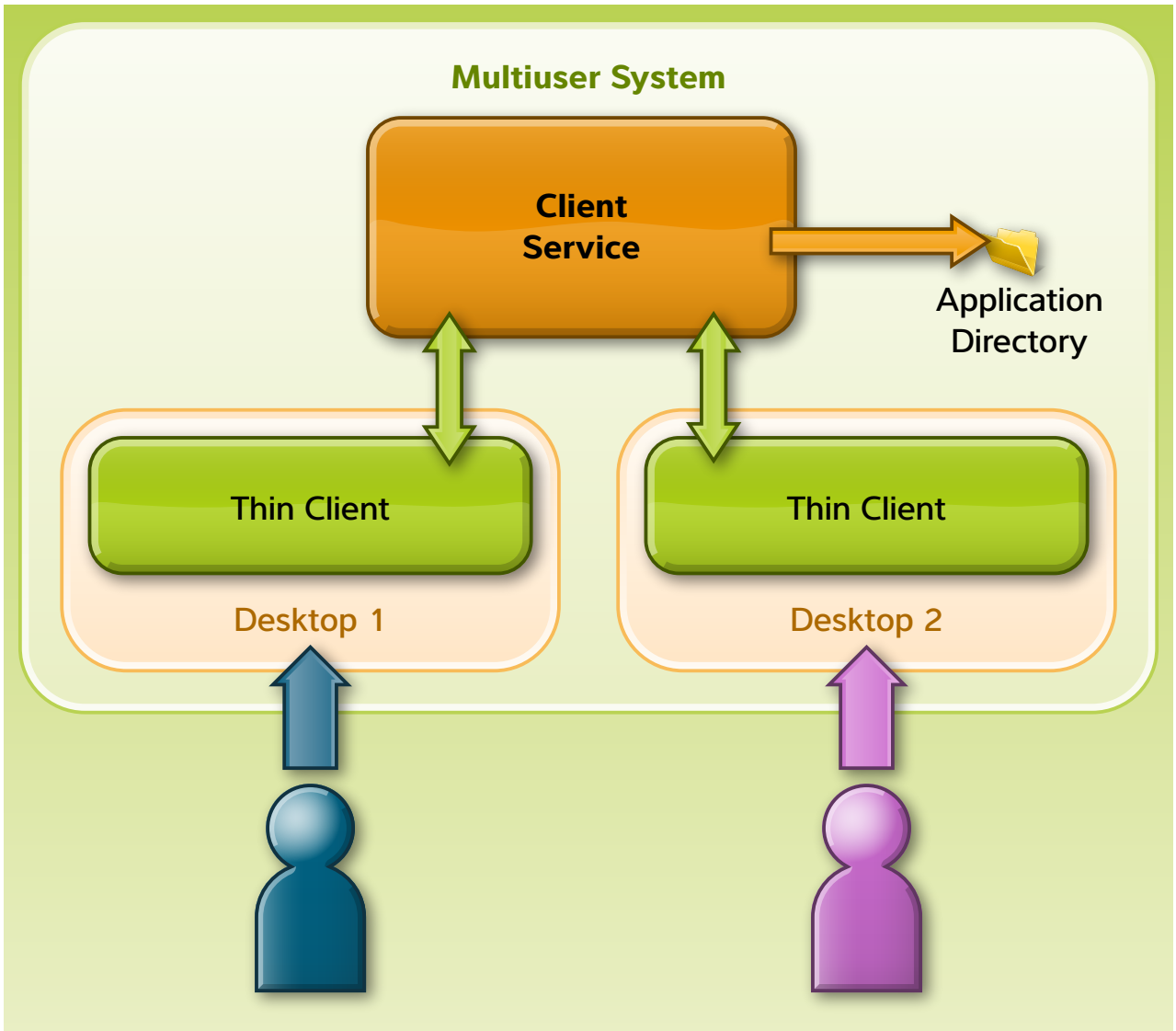


Illustration 5 An overview how the thin client and client service system is working.

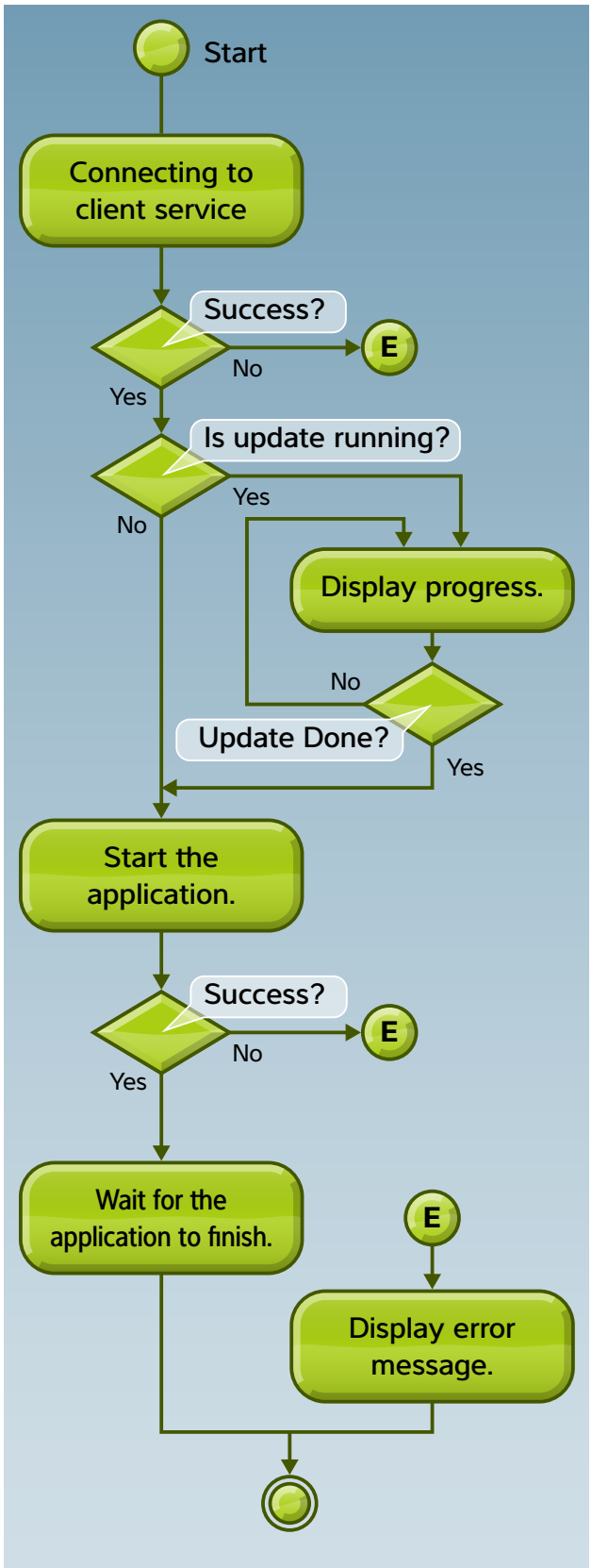
- **Automatic termination of the application on update:** If there is an update and a running application is blocking the update process, the application is terminated after a grace period. A message is displayed to the user, demanding he closes the application in a given time period.
- **Update notification:** For applications which do not block files or directories in the application directory, the update is done silently in the background. After the update a message for all users which currently running the updated application is displayed to request closing and re-opening the application.

The combination of the thin client and client service is therefore the ideal solution on multiuser systems.

See Illustration 5 on page 35 The client service is running on the multiuser system with elevated privileges to be able to write to the application directory. Each user has a own desktop on the system and is running the thin client on start of the desired application. The thin client connects to the running client service to check if there is an update in progress or if the application can be started.

After receiving the clearance from the client service, the thin client is starting the application with the rights of the user.

Mode of Operation



The following sections describe the mode of operation of the thin client.

Start-Up

After the start, the thin client tries to connect with the running client service on the system. It is using the configured port to connect to the local host and establish a secure *SSL* connection. If there is any error to connect to the service, an error message is displayed and the thin client stops.

The thin client is sending data about the user and context to the service and requests the status of the specified project. If the chosen project does not exist, an error message is displayed and the thin client stops.

If there is an update in progress, the current status and progress of the update is displayed. See Screen 1 on page 38 for an example how the connect and progress dialogue looks like.

In the case an update is running, the current progress of the update is displayed in the dialogue. The thin client is waiting until an update is done and the application is up to date.

Application Start

If the application is up to date, it is started in the same context as the thin client was started. If the start of the application fails, an error message with the reason for the failure is displayed.

The successful start is reported to the client service.

Monitoring the Process

The running process is monitored while it is running. The thin client keeps the connection open

Screen 1: The connect and progress dialogue of the thin client.

to the client service while the process is running to get notifications about an update or problem with the process.

If the process finishes, the end of the process is reported to the client service and the connection is closed. This happens if the process quits normal or it is crashing. After closing the connection the thin client stops.

Notifications

There are two kind of notifications. The first is just an informal notification about the update of the application. The second is sent if the running process is blocking important files which are preventing a running update.

Update Notification

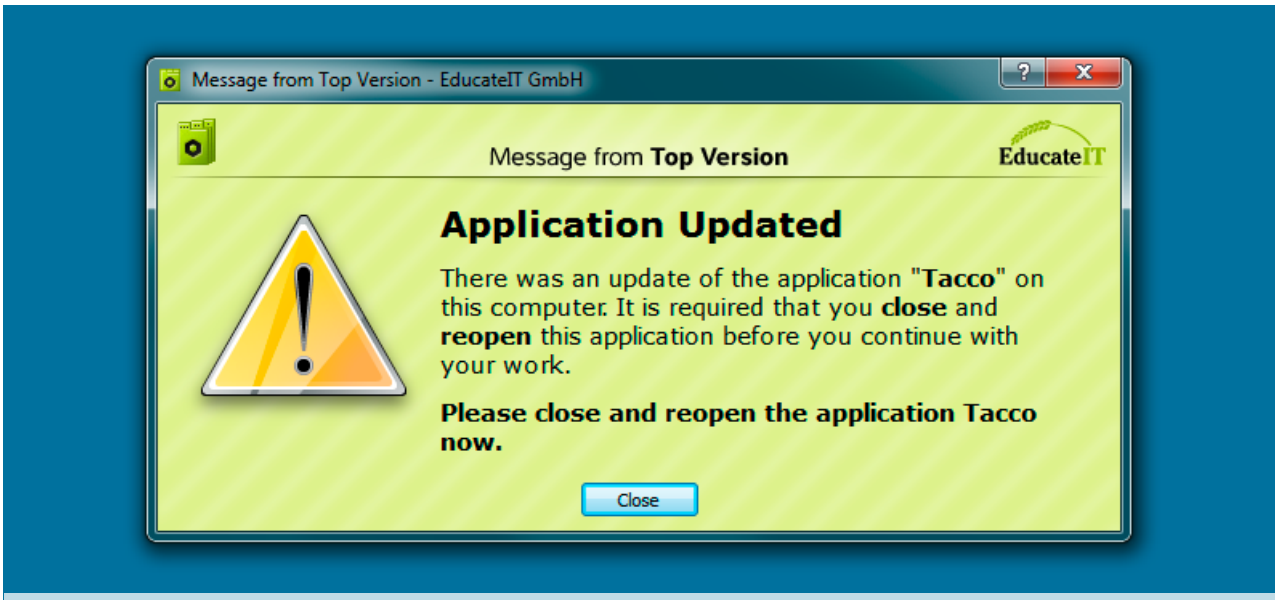
Screen 2 on page 39 shows the informal update notification. It is shown if the application could get updated even if a process of the application is currently running.

The dialogue reminds the user to close and re-open the application as soon as possible to start the new version.

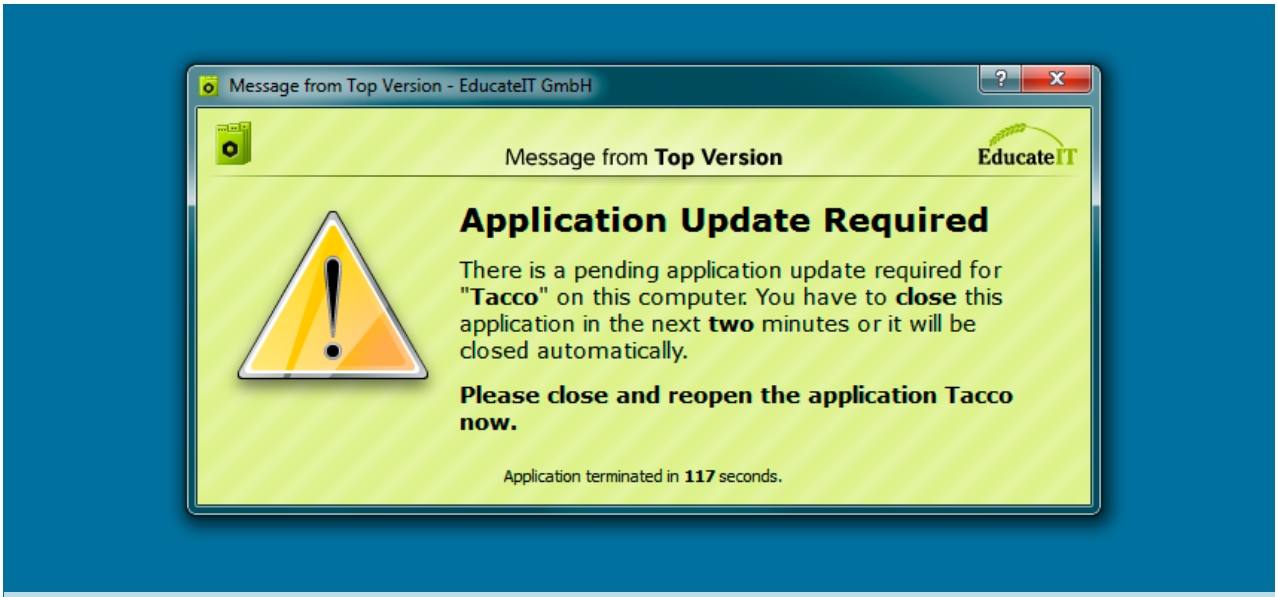
Application Close Request

Screen 3 on page 39 is showing the close request sent if the running process is blocking files which are preventing a successful update.

At the bottom of the dialogue, a count down with the remaining time to close the application is shown. If the time runs out, the process of the application is closed automatically.



Screen 2: Notification about an application update.



Screen 3: Notification which requests closing the application.

Application Termination

If an application process is blocking important files which need an update, the application is automatically terminated by the thin client after a given time-out.

First a `SIGTERM` signal is sent to the application. This allows the application to close properly. If the application is still running after 10 seconds, a `SIGKILL` signal is sent to the application. This terminates the process in any case.

Command Line Arguments

The thin client has no file based configuration. Everything is passed with command line arguments to the client.

Table 2 on page 41 is showing all available command line arguments and their meanings and Listing 9 on page 41 shows a few ways how to start the thin client.

Argument	Description
`--applicationId` `-a`	With this required parameter you specify the identifier of the application to start. This identifier is defined in the configuration of the client service. The identifier is case sensitive.
`--serverPort` `-p`	This optional parameter changes the server port which is used to connect to the local service. By default port 19693 is used.

Table 2 Command line arguments for the thin client.

```
TopVersionThinClient.exe --applicationId=example1
TopVersionThinClient.exe -a=test2
TopVersionThinClient.exe --applicationId=example3 --serverPort=4444
TopVersionThinClient.exe -a=test4 -p=4444
```

Listing 9 Examples how to start the thin client.

Chapter 4: Client Service

This chapter describes the client service which provides automatic updates on a multiuser system and acts as back-end for the thin client.

Preface

The *client service* can be used alone or in combination with the thin client which is described in chapter "Thin Client" on page 33. It is an alternative to the *normal client*, originally designed for multiuser systems.

There are a number of features which are only provided by the *thin client* with the *client service* combination. See "Preface" on page 35 for the features of the *thin client*. In addition, the service provides this features:

- **Automatic update check:** You can configure a given interval in which the service checks for updates of the application. If an update is found, it is installed automatically.
- **Smart handling of blocked/locked files:** Even if there are some locked or blocked files, caused by running applications, the smart update algorithm can nevertheless install the update. Locked files are just moved into some temporary folder which is removed after a given time.
- **Automatically clean-up:** Locked files are just moved out of the way until the lock is removed. A smart clean-up algorithm is removing these temporary folders after a few days automatically to free hard disk space.
- **EducateIT Service Framework:** The service is using our own powerful service framework and is supporting all of the features of this framework.
- **EducateIT Log System:** The service is also using our log system and is supporting all features of it.

The combination of the *thin client* and *client service* is therefore the ideal solution on multiuser systems.

You should got the "*Log System – Administrator Manual*" and the "*Service Framework – Administrator Manual*" along with this documentation. These two documents are describing all these additional features in detail.

Configuration

The client service is using the "*Configuration B*" system for the configuration. You should get a manual titled "*Configuration B – Administrator Manual*" along with this document. It describes all the basics and possibilities of the configuration system and if you have not already read it, you should read it now.

Illustration 6 on page 48 shows the structure and all the values of the configuration for the *client service*. They are described in the following sections in detail.

There is also an example configuration in Listing 10 on page 49 which defines one managed application and also contains a simple log configuration entry.

ServerInterface

This optional value controls on which *interface* the service listens for incoming connections. This has to be a valid *IP address* which is assigned to an interface of the server.

By default the service is just listening for "*localhost*" connections which is perfect for a productive environment. I recommend to omit this value and use the default.

ServerPort

This optional value changes the port on which the service listens for incoming connections. Define this value if there is a conflict with another service on the same system.

The default port which is used if you omit this value is `19693`. For productive systems I recommend to omit this value and use the default port number except there is a local conflict.

ApplicationList

The application list defines one or more local applications which shall be managed by the client service. Each entry needs a unique *application identifier* value. This value is used by the *thin client* to identify the application which shall get started. The values of each entry are described in the following sections.

Module — `Application`

List — `ApplicationList`

Constraints
- Must not be empty.

Definition — List Entry Definition

Value — `AllowAutomaticDowngrade` — Boolean

Value — `Channel` — String

Constraints
- Must not be empty.
- Minimum length: 1 character(s).
- Maximum length: 64 character(s).
- Has to match the following **case insensitive** regular expression: `[a-z][a-z0-9]{0,63}`

Value — `Identifier` — String

Constraints
- Must not be empty.
- Minimum length: 1 character(s).
- Maximum length: 64 character(s).
- Has to match the following regular expression: `\w+`

Value — `Project` — String

Constraints
- Must not be empty.
- Minimum length: 1 character(s).
- Maximum length: 64 character(s).
- Has to match the following **case insensitive** regular expression: `[a-z][a-z0-9_]{0,63}`

Value — `Repository` — String

Constraints
- Must not be empty.

Value — `StartWithErrors` — Boolean

Value — `UpdateCheck` — String List

Constraints
- Each element of in this list has to be one of the following values: `ByUser`, `EveryMinute`, `EveryHour`
- Must not be empty.

Value — `ServerInterface` — Optional — String

Illustration 6 The visualization of the `Application` configuration module of the client service.

Constraints	▪ Must be a valid host address.
Value	`ServerPort` Optional Integer
Constraints	▪ Minimum value: 1 ▪ Maximum value: 65535
Default	19693

```xml
<?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="ServerInterface">xxxx</Value> -->
    <!-- <Value name="ServerPort">yyy</Value> -->

    <List name="ApplicationList">
      <ListEntry>
        <Value name="Identifier">projecta</Value>
        <Value name="Repository">//server01/TopVersion/ProjectA/repository.xml</Value>
        <Value name="Project">Project_A</Value>
        <Value name="Channel">Production</Value>
        <Value name="StartWithErrors">No</Value>
        <Value name="AllowAutomaticDowngrade">Yes</Value>
        <Value name="UpdateCheck">ByUser EveryMinute</Value>
      </ListEntry>
    </List>

  </Module>

  <Module name="Log">
    <List name="FileWriter">
      <ListEntry>
        <Value name="LogFilePath">E:/TopVersion/TopVersionClientService.log</Value>
      </ListEntry>
    </List>
  </Module>

</Configuration>
```

Listing 10 Example configuration for the *Top Version* client service.

ApplicationList » Identifier

The identifier for each entry in the application list has to be unique. It is used by the client to identify the application to start. The identifier has be shorter than 64 characters and only contain letters, numbers and the underscore character.

There is no relation between this identifier and the repository. The sole purpose of this identifier is to identify the right application entry for the client. It is also used in the log files.

The following block contains some examples for valid identifier:

```
projecta
App1
Project_A
app20
```

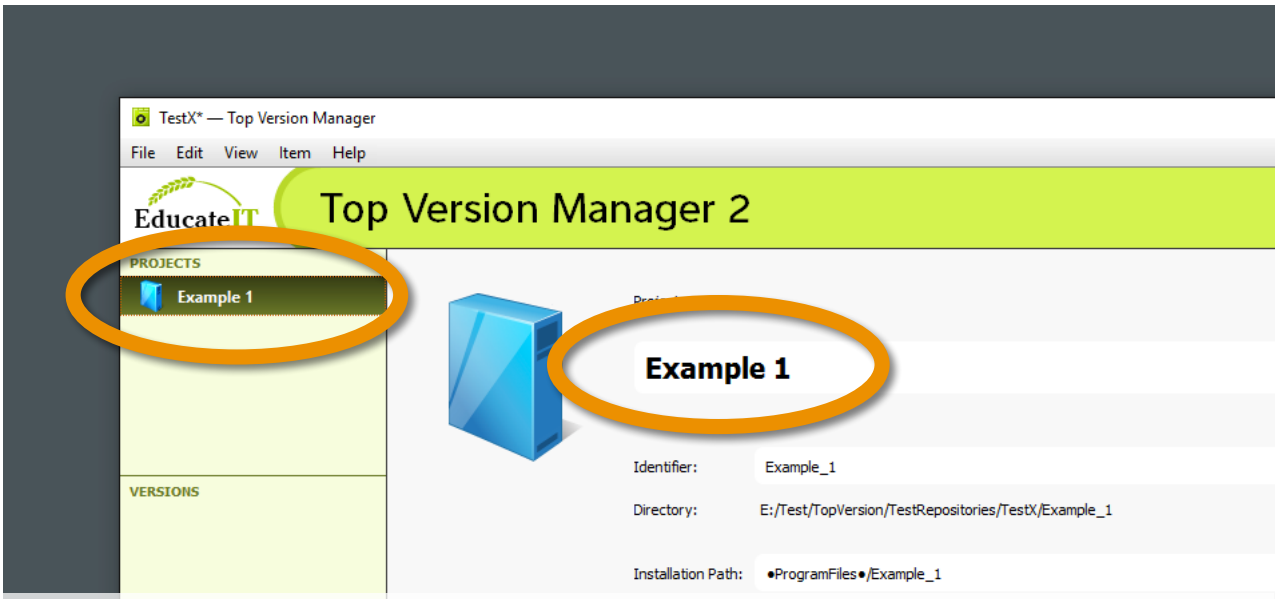
ApplicationList » Repository

This is the absolute path to the repository XML file in which the project of the application is hosted. Some example paths are shown in the following block:

```
E:/Test/TopVersion/Test/repository.xml
C:/Repro/repository.xml
//server01/TopVersion/ProjectA/repository.xml
```

ApplicationList » Project

This value has to be set to the project name from the repository. It has to be the exact project name, which is case sensitive. See



Screen 4: The project name in the repository manager.

Screen 4 on page 50 where to find the project name in the repository manager.

ApplicationList » Channel

This value has to bet set to the chosen channel name for the selected project. It has to be the exact channel name which is case sensitive. See Screen 5 on page 51 where to find the channel name in the repository manager.

ApplicationList » StartWithErrors

This flag controls if the application should be started if there were errors updating the application to the latest requested version.

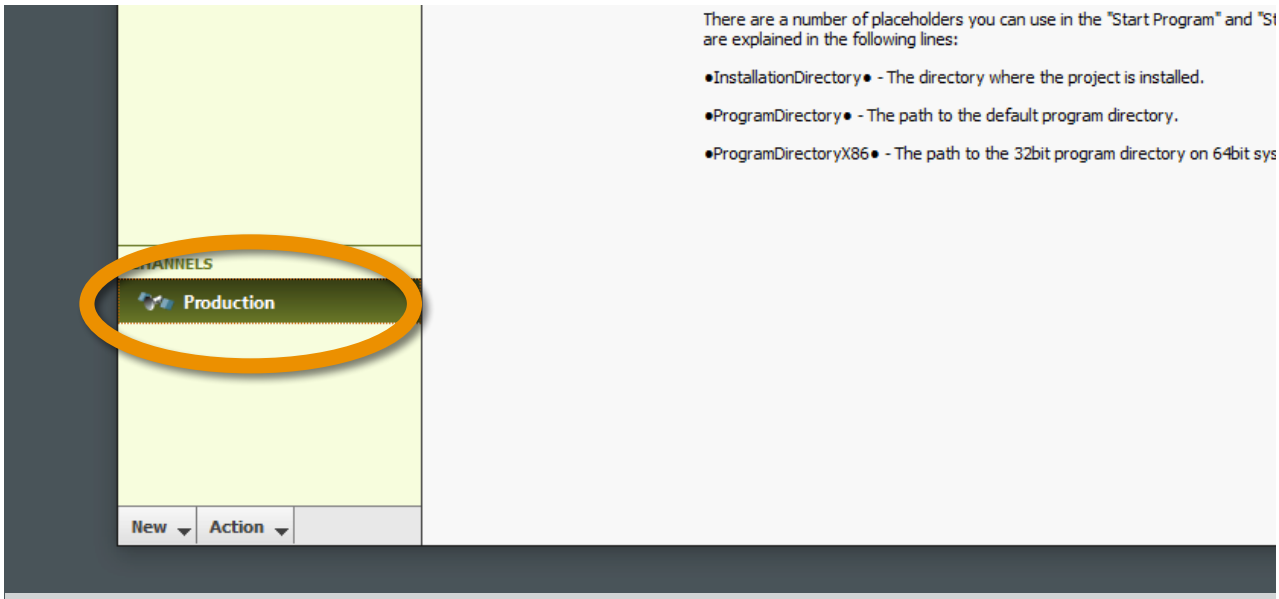
If this flag is set to `Yes`, the application is started nevertheless its an older version because of a failed update.

ApplicationList » AllowAutomaticDowngrade

This flag controls if an automatic downgrade of the application is allowed. If this flag is set to `Yes`, an downgrade is made otherwise the update is stopped with an error message in the log.

ApplicationList » UpdateCheck

This value contains a number of *tags* which control the condition when an update is started. Table 3 on page 52 describes all valid *tags*. The *tags* are *case sensitive* and have to be separated with *whitespace*.



Screen 5: The channel name in the repository manager.

The following box contains some common combination of tags for this value:

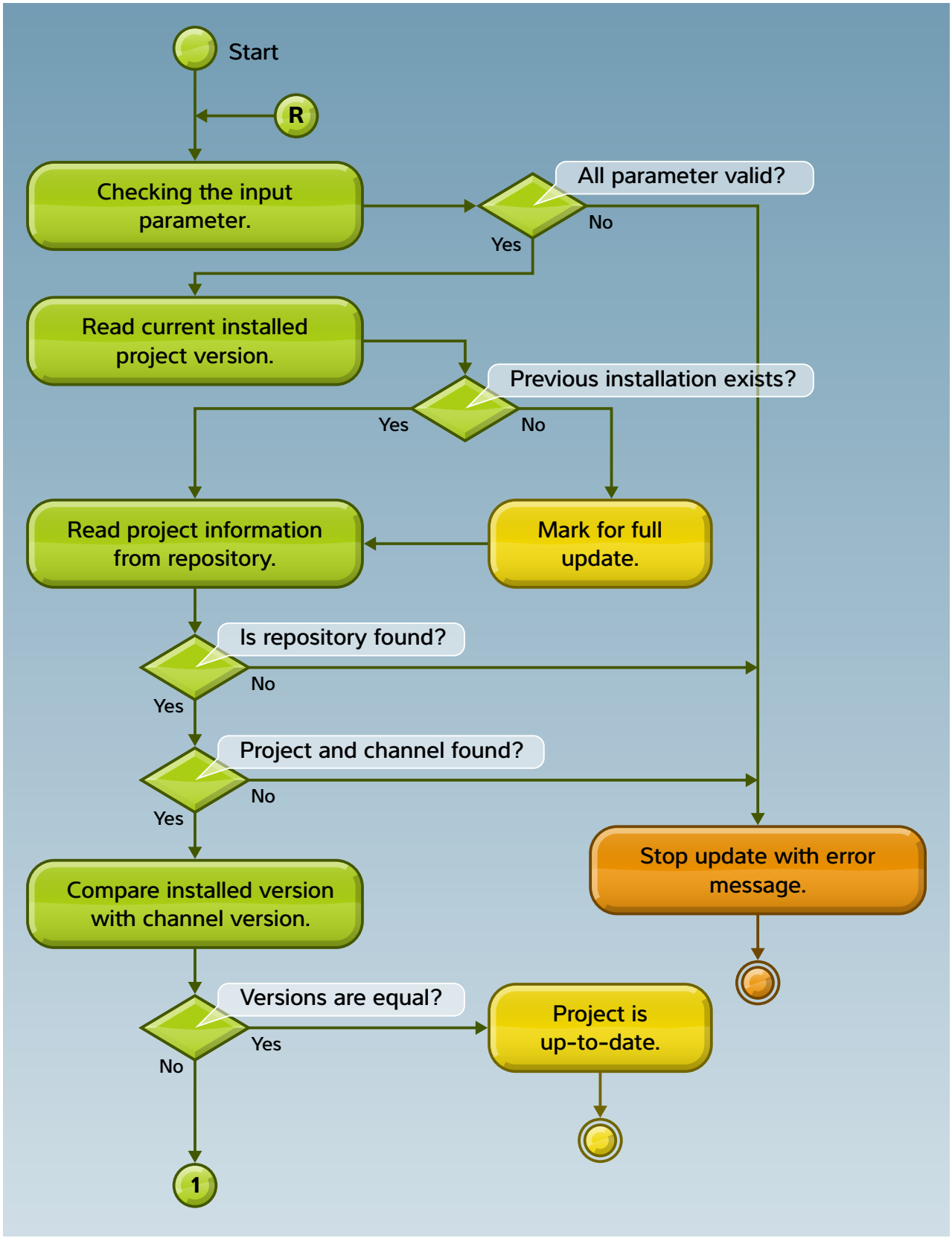
```
ByUser
ByUser EveryMinute
ByUser EveryHour
EveryMinute
EveryHour
```

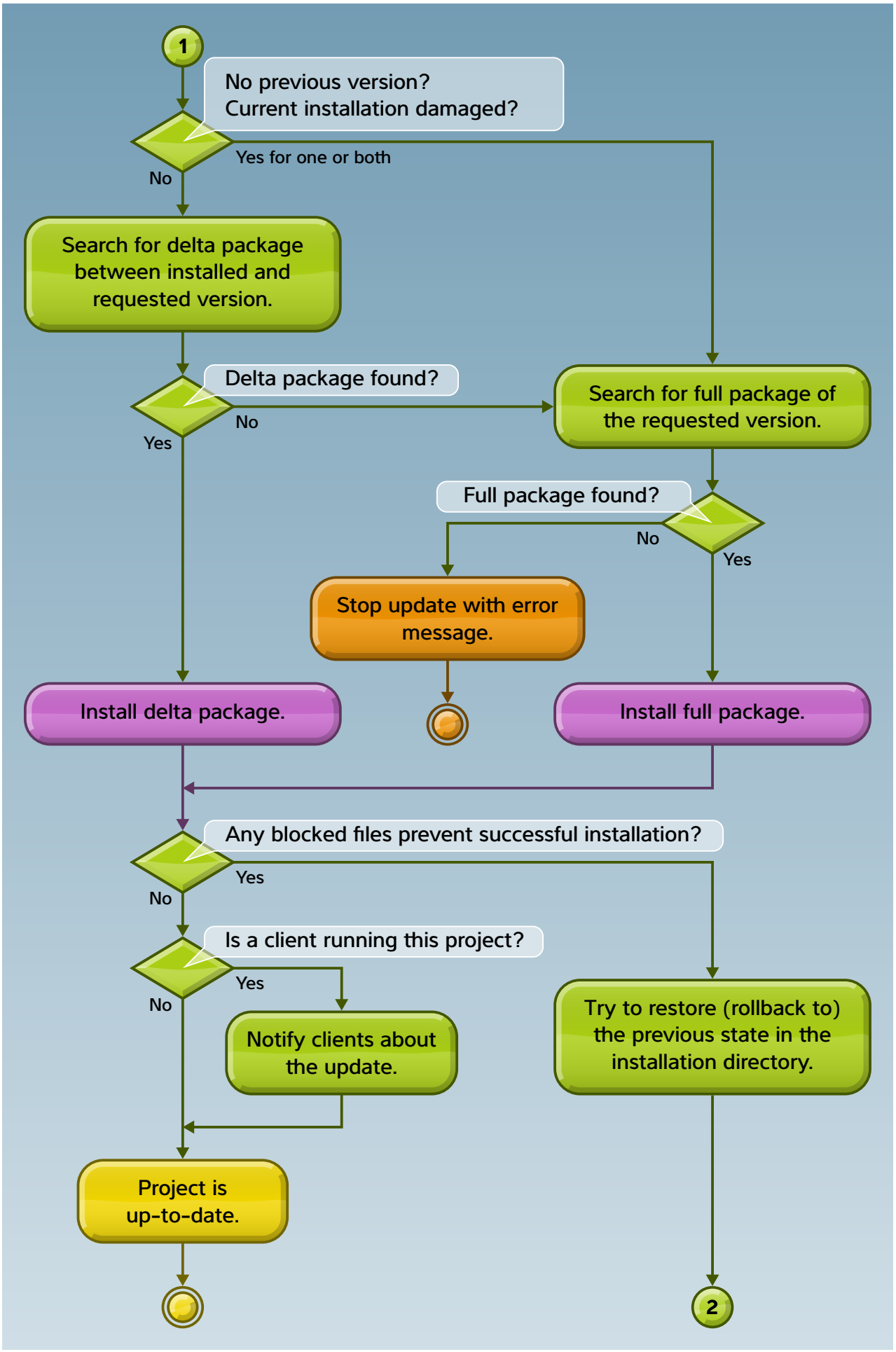
If there are multiple tags specified, the time of the update check is determined by the simple rule: which event comes first. And if there is already an update check running at the given time, this event is ignored.

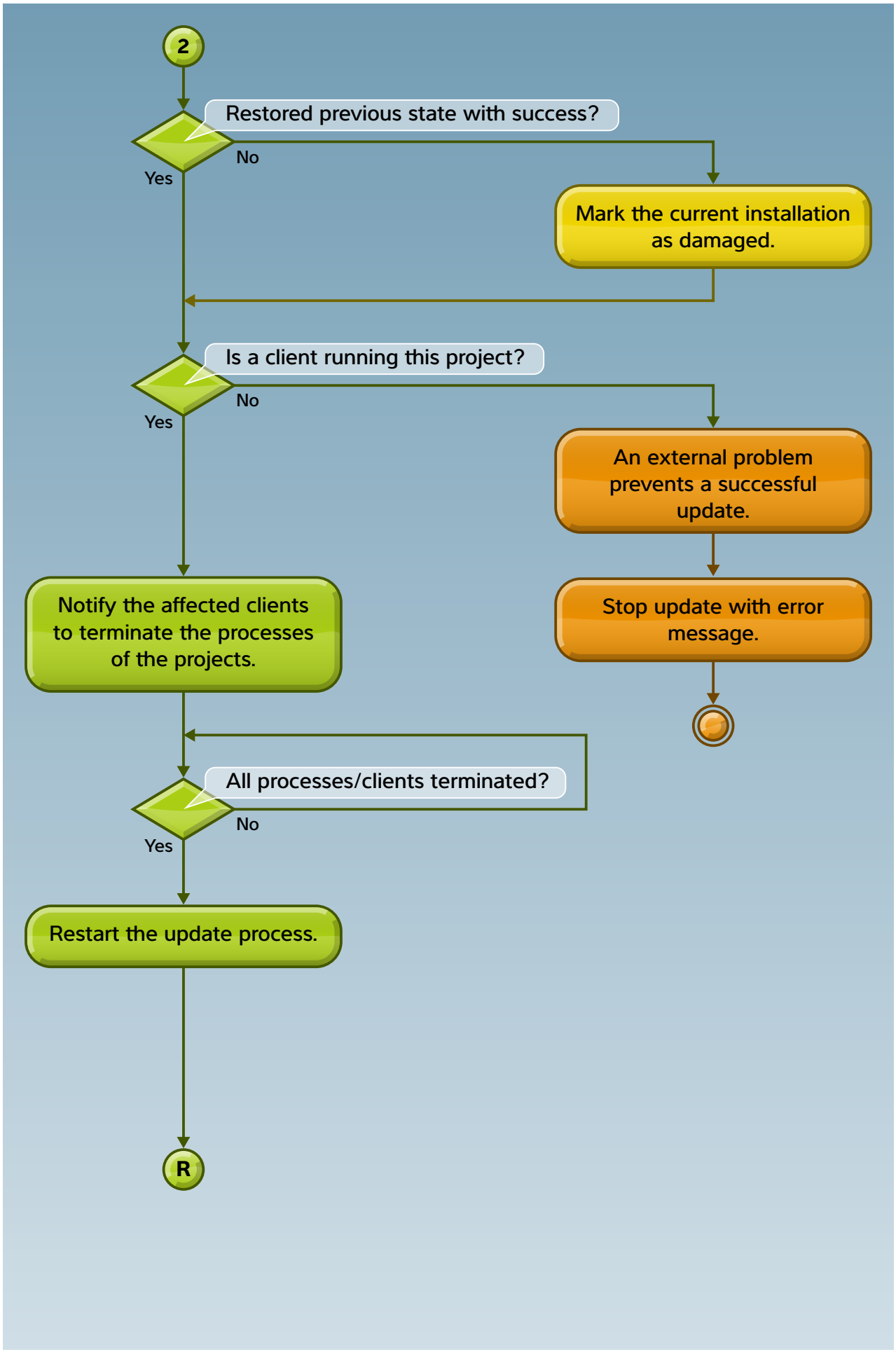
Tag	Description
`ByUser`	If this tag is set an update check is always triggered if a user is starting an application using the thin client.
`EveryMinute`	If this tag is specified, an update check is done every minute. This tag has priority over the `EveryHour` tag. In the case a `EveryMinute` and a `EveryHour` tag are present, the `EveryMinute` tag has precedence.
`EveryHour`	If this tag is present, an update check is done every hour.

Table 3 All valid tags for the `UpdateCheck` value.

Update: Mode of Operation

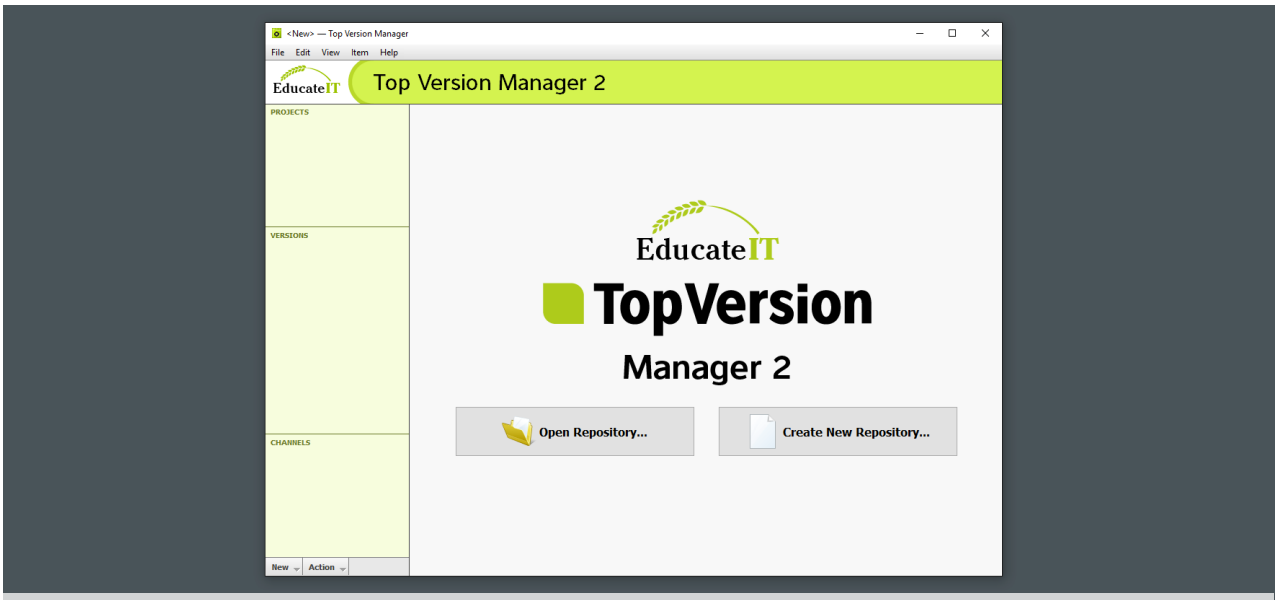



1
No previous version?
Current installation damaged?
Yes for one or both
No
Search for delta package between installed and requested version.
Delta package found?
No
Yes
Search for full package of the requested version.
Full package found?
No
Yes
Stop update with error message.
Install delta package.
Install full package.
Any blocked files prevent successful installation?
Yes
No
Is a client running this project?
Yes
No
Notify clients about the update.
Try to restore (rollback to) the previous state in the installation directory.
Project is up-to-date.
2

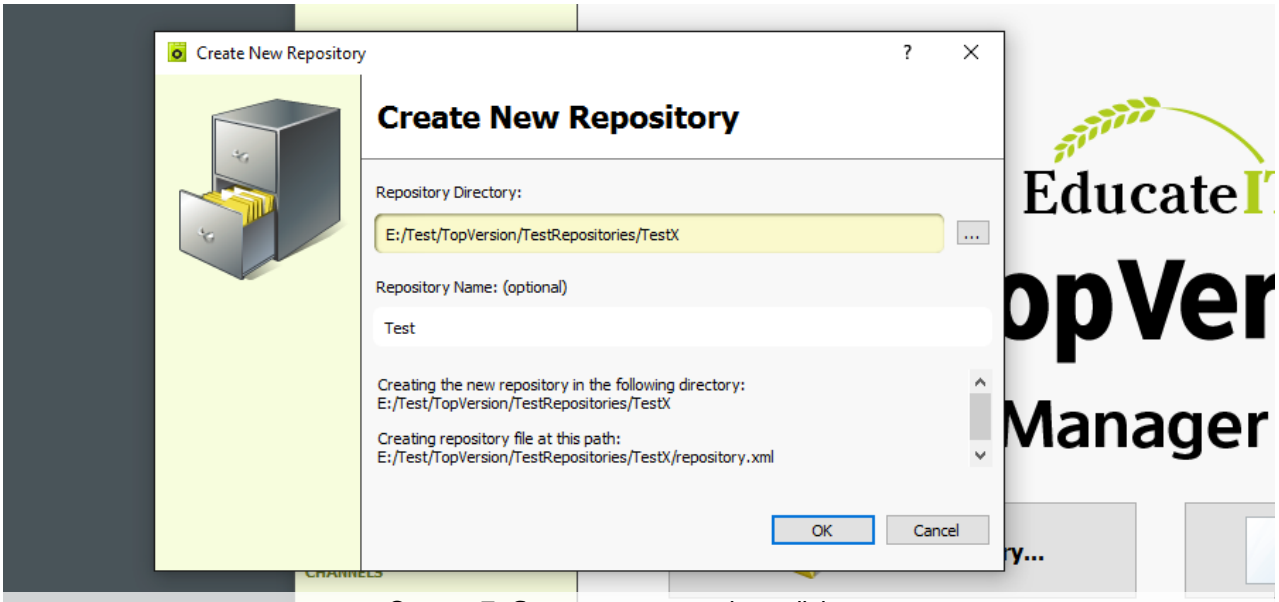

2
Restored previous state with success?
No
Yes
Mark the current installation as damaged.
Is a client running this project?
No
Yes
An external problem prevents a successful update.
Notify the affected clients to terminate the processes of the projects.
Stop update with error message.
All processes/clients terminated?
No
Yes
Restart the update process.
R

Chapter 5: Manager

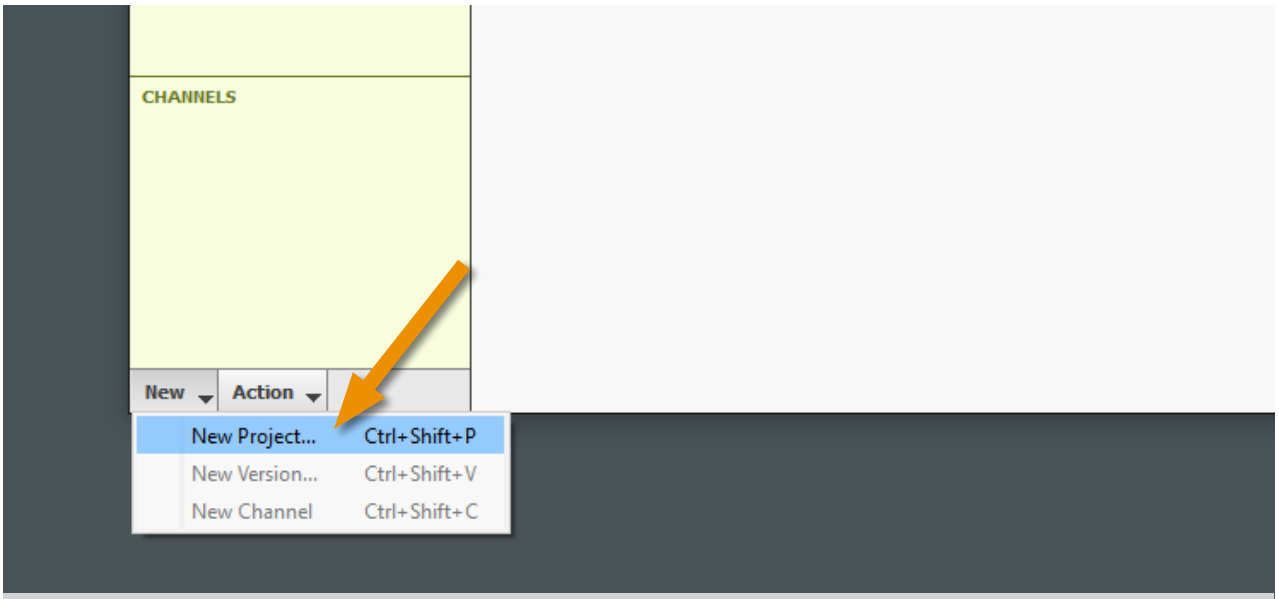
This chapter describes the *Top Version* manager.



Screen 6: The welcome page.



Screen 7: Create a new repository dialogue.



Screen 8: 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 6 on page 58). Here you have the choice to open an existing repository or to create a new one.

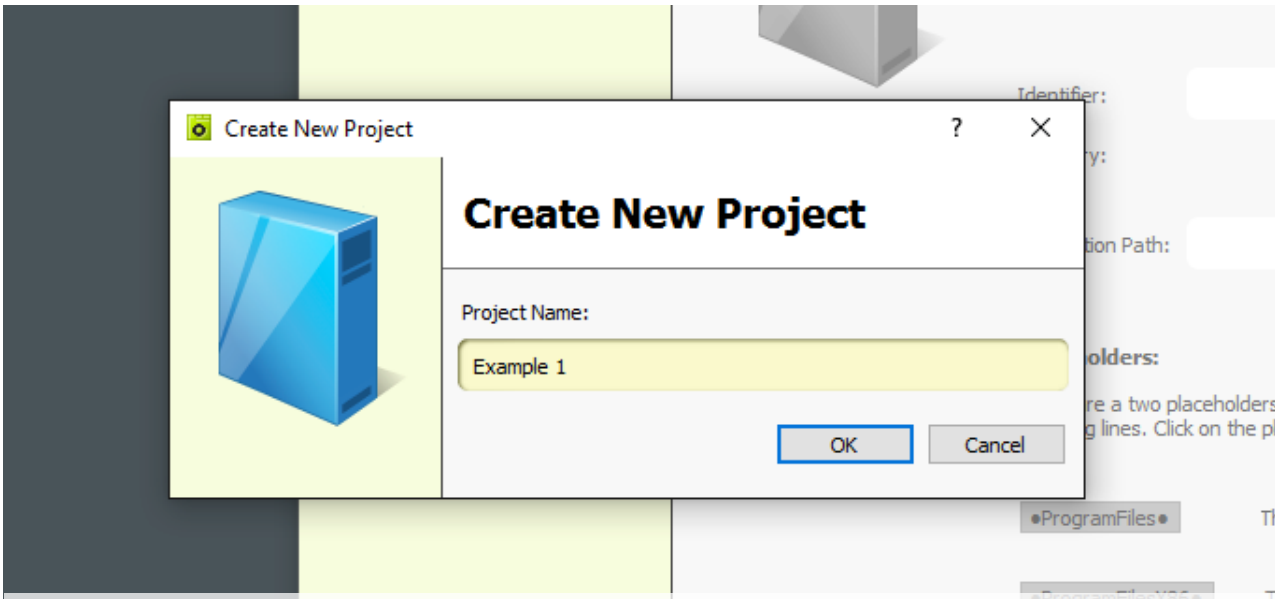
Click on the button labelled “Create New Repository...” to create a new repository. A dialogue appears (Screen 7 on page 58) where you have to choose the path to the new repository. This has to be an empty folder somewhere.

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.

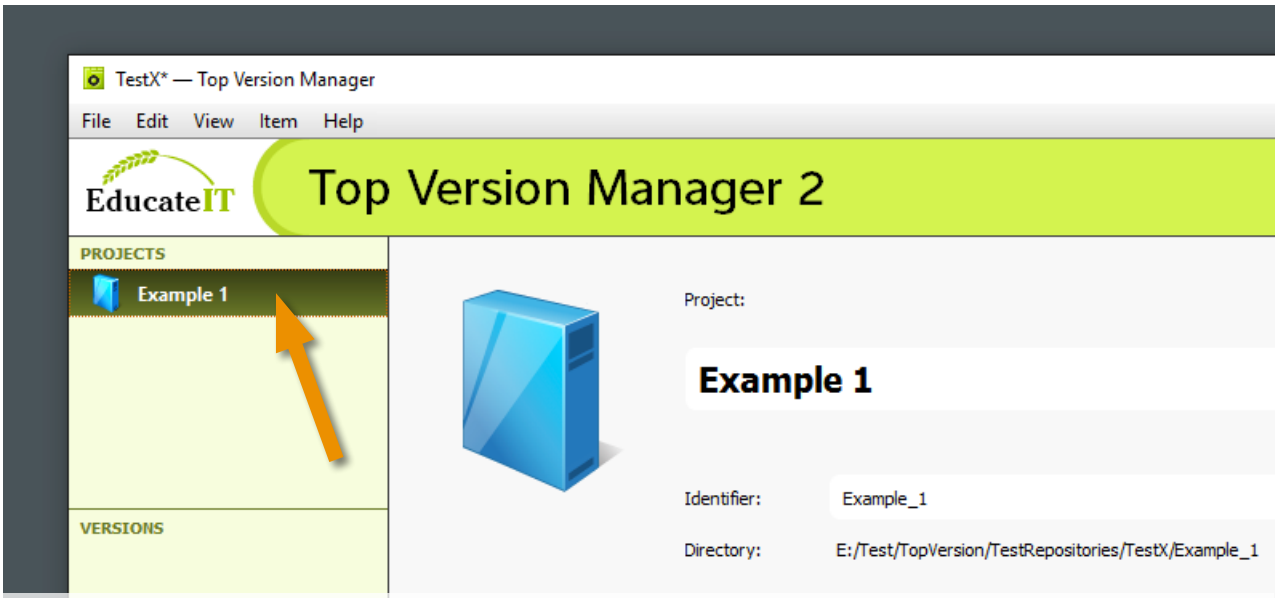
Create a New Project

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 8 on page 58).

A new dialogue appears (Screen 9 on page 59) 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.

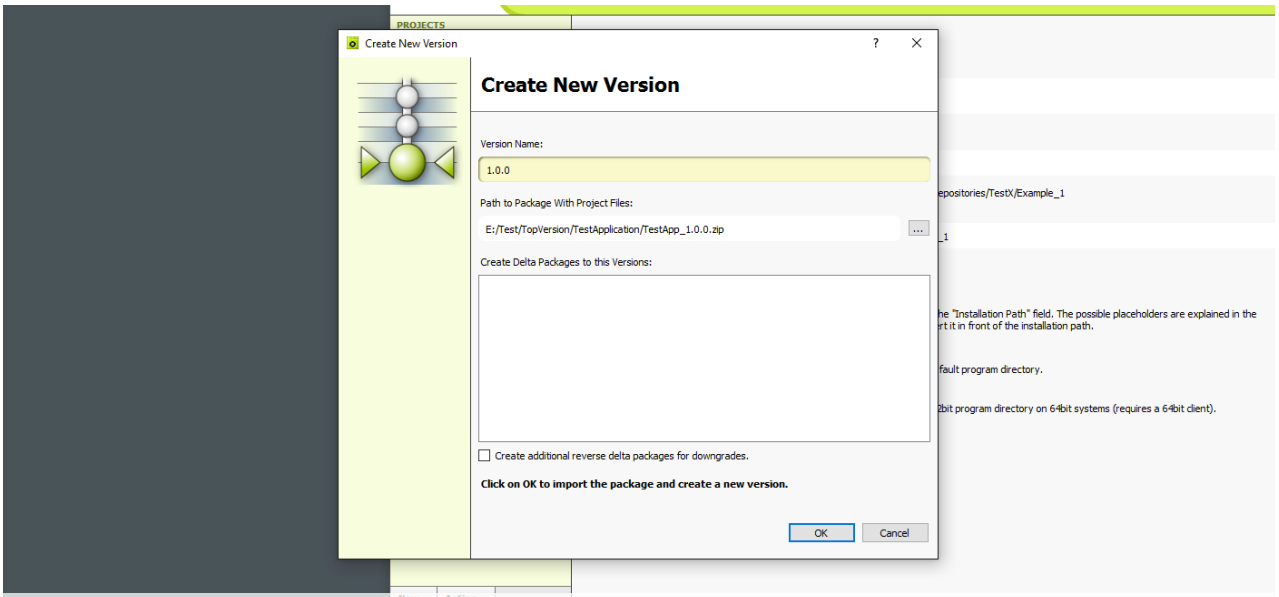


Screen 9: The dialogue to enter the name for the new project.



Screen 10: Click the project on the left side to see the details.

Screen 11: While the project is not saved, you can edit the project identifier.



Screen 12: The dialogue to create a new version.

Clicking on “OK” will create the new project.

The new project is shown in the project list on the left side of the window (Screen 10 on page 60). 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 11 on page 60). Once you save the project, the identifier can not edited anymore.

Create a New Version

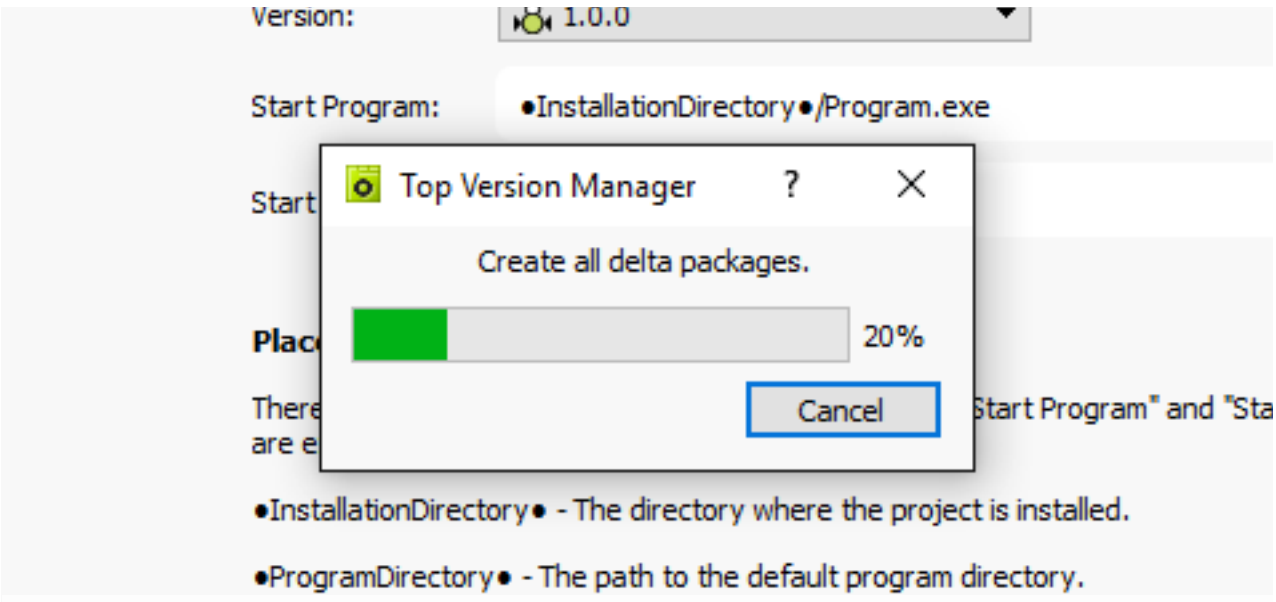
To create a new version you can use the “New” drop down in the bottom left corner of the window. Select “New Version...” in the pop-up menu. All commands are also available in the application menu.

A new dialogue opens (Screen 12 on page 60) 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 13 on page 61). 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 problem message is shown. The repository is not changed in this case and you can fix the problem and try again.



Screen 13: 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.

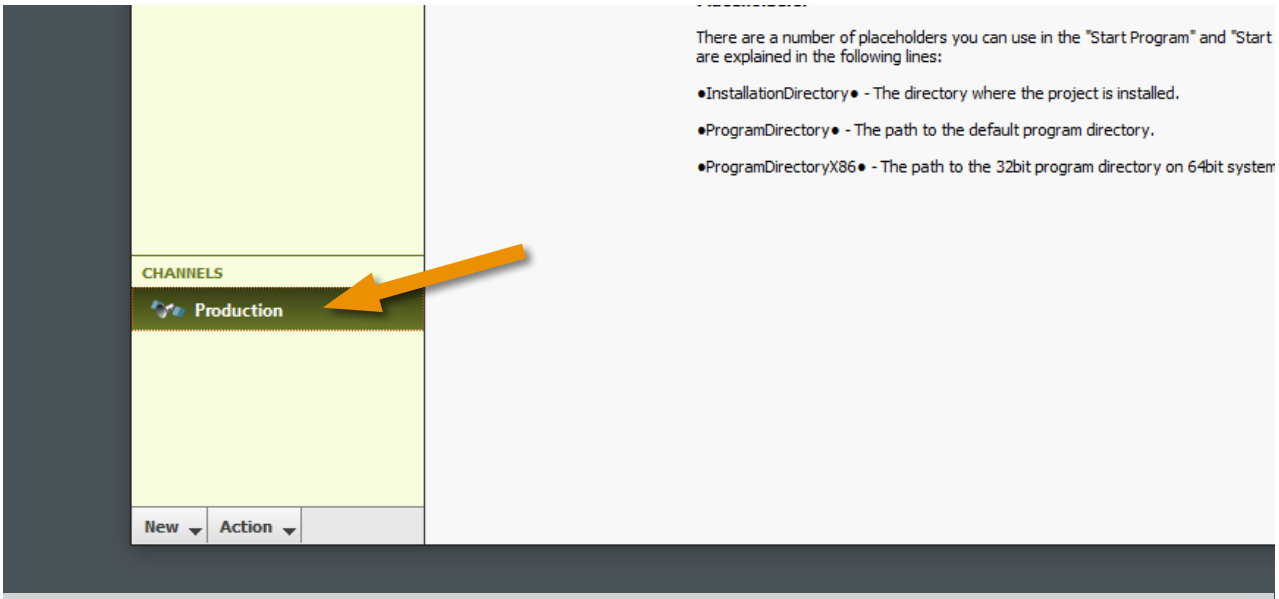
Use the “New” drop down in the bottom left corner and select “New Channel...”. A new channel is added, using one of the predefined names (Screen 14 on page 62).

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.



Screen 14: The new channel is added with one of the predefined names.

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.

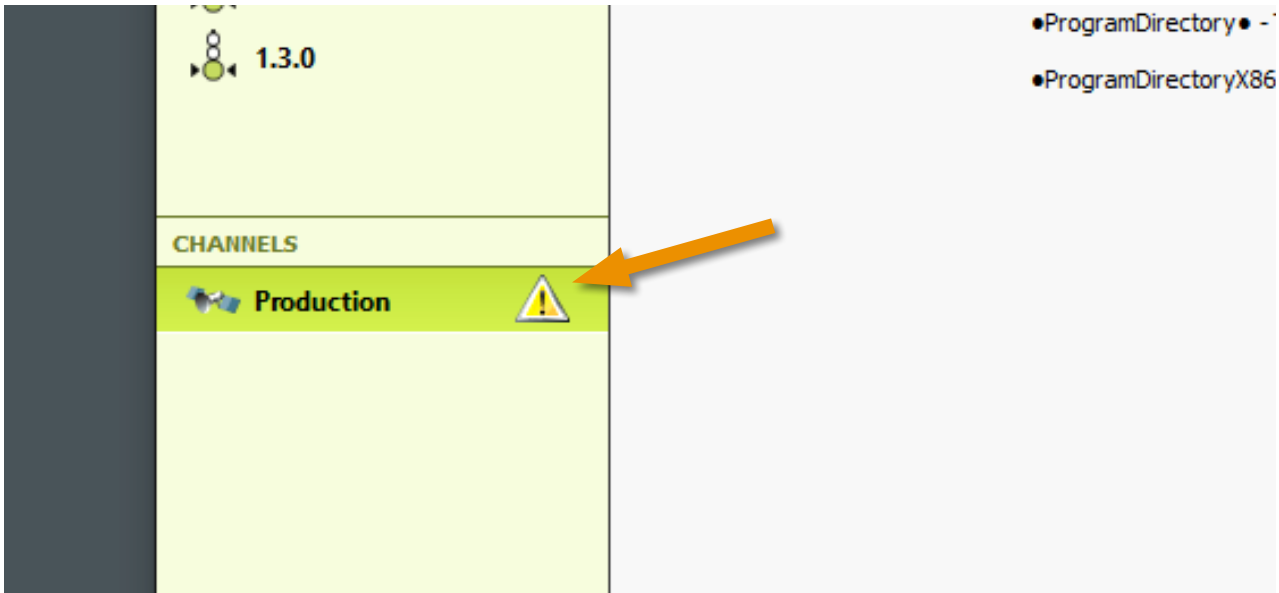
See Screen 15 on page 63 how the warn sign is displayed.

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

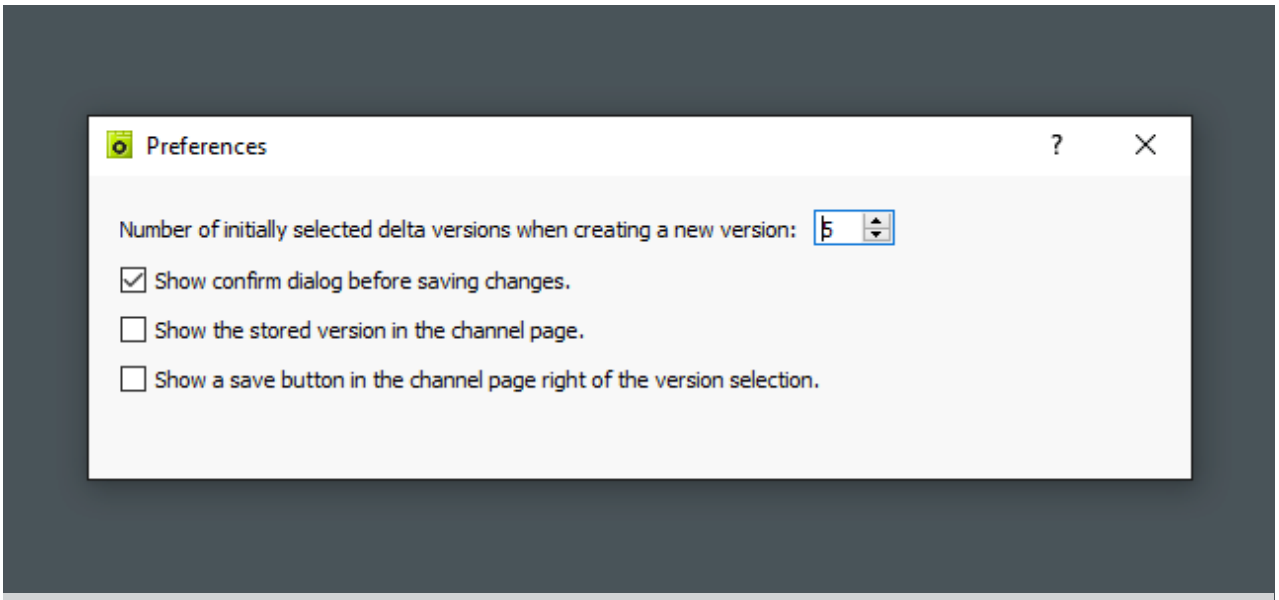
Preferences

There are some features which can only be enabled or disabled in the preferences of the user interface. See Screen 16 on page 64 for the preferences window.

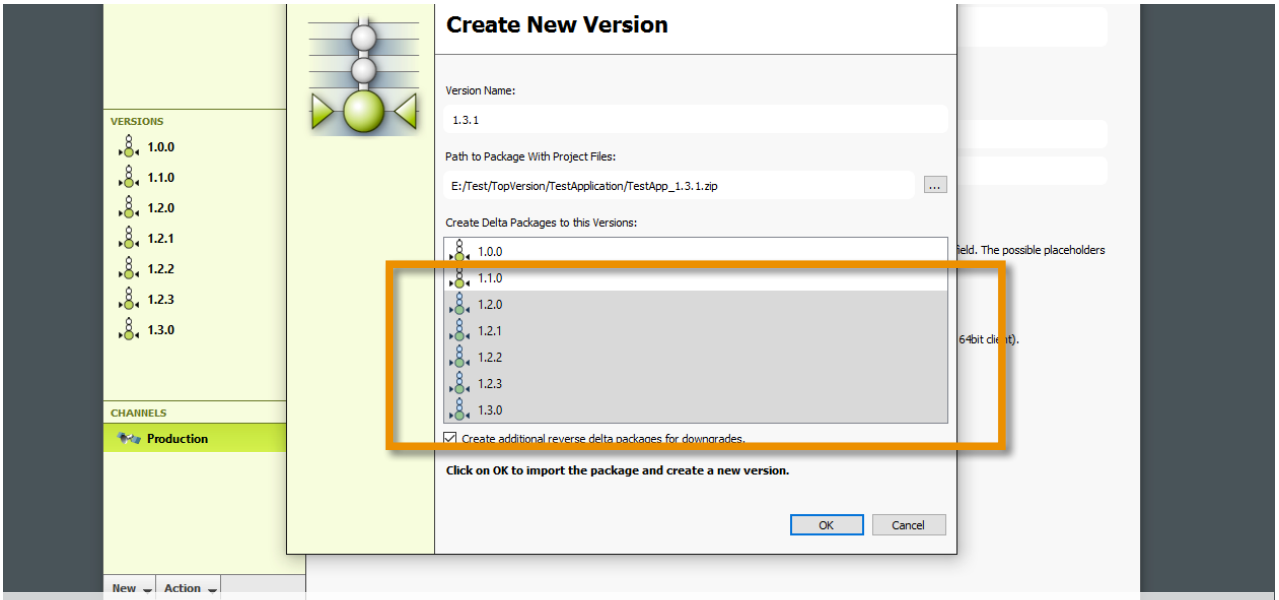
The entries in this preferences window are described in the following sections.



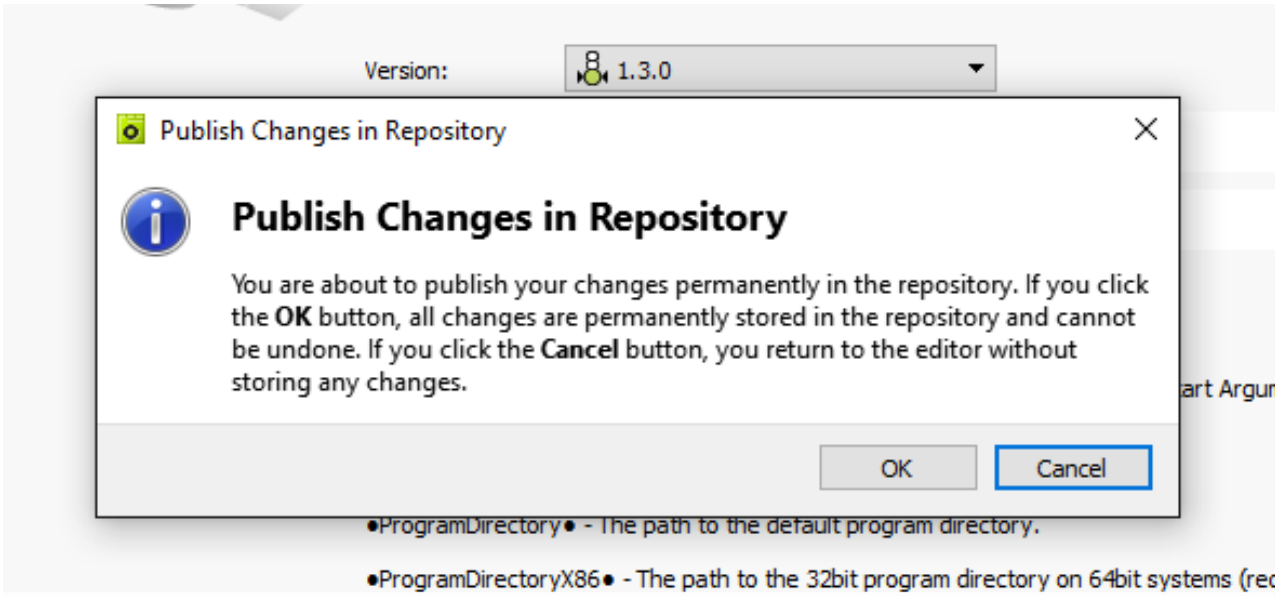
Screen 15: If you change the version in a channel, a warn sign appears.



Screen 16: The window with the preferences for the user interface.



Screen 17: You can change the number of automatically selected delta versions in the preferences window.



Screen 18: 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 17 on page 64 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

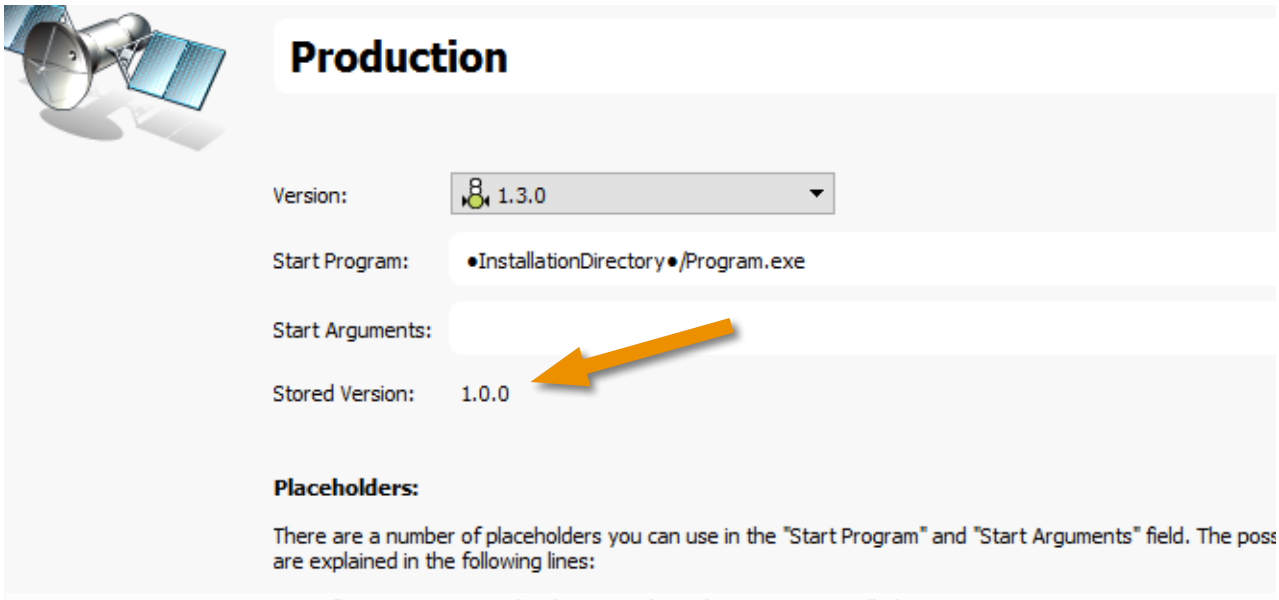
When this check box is checked, a confirmation dialogue is shown when you save your changes. See Screen 18 on page 64 for the confirmation dialogue. You have to confirm, that you like to save all made changes into the repository. At this point you have the choice to cancel, and leave the repository untouched.

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 19 on page 65 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.



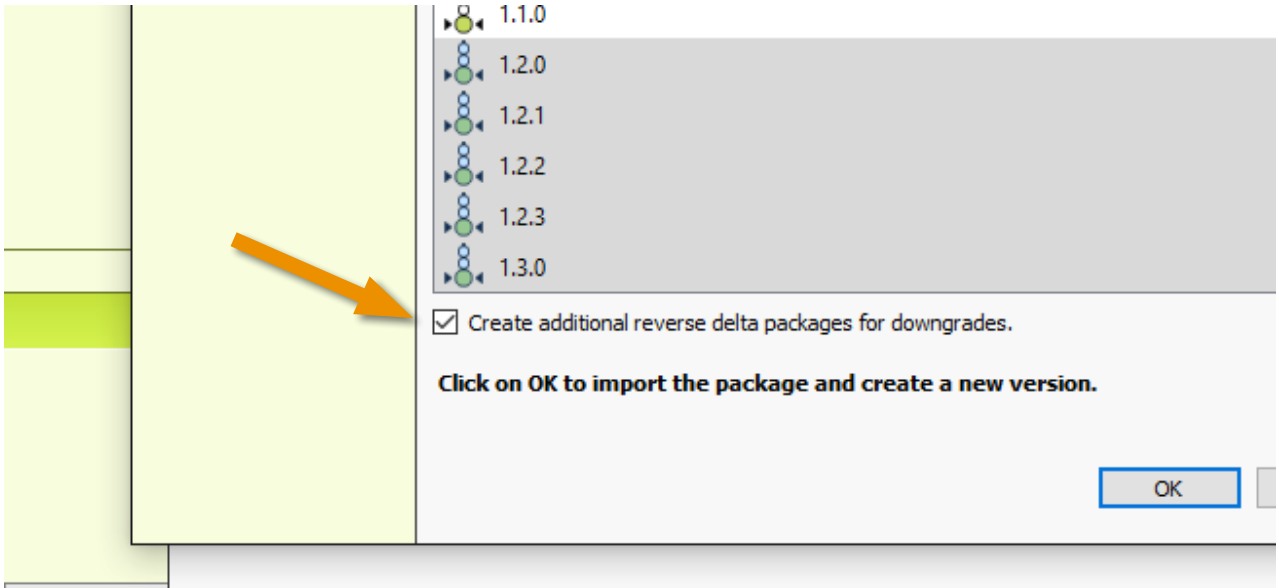
Screen 19: Show the stored version number in the channel page.

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 20 on page 67 where this check box is checked.



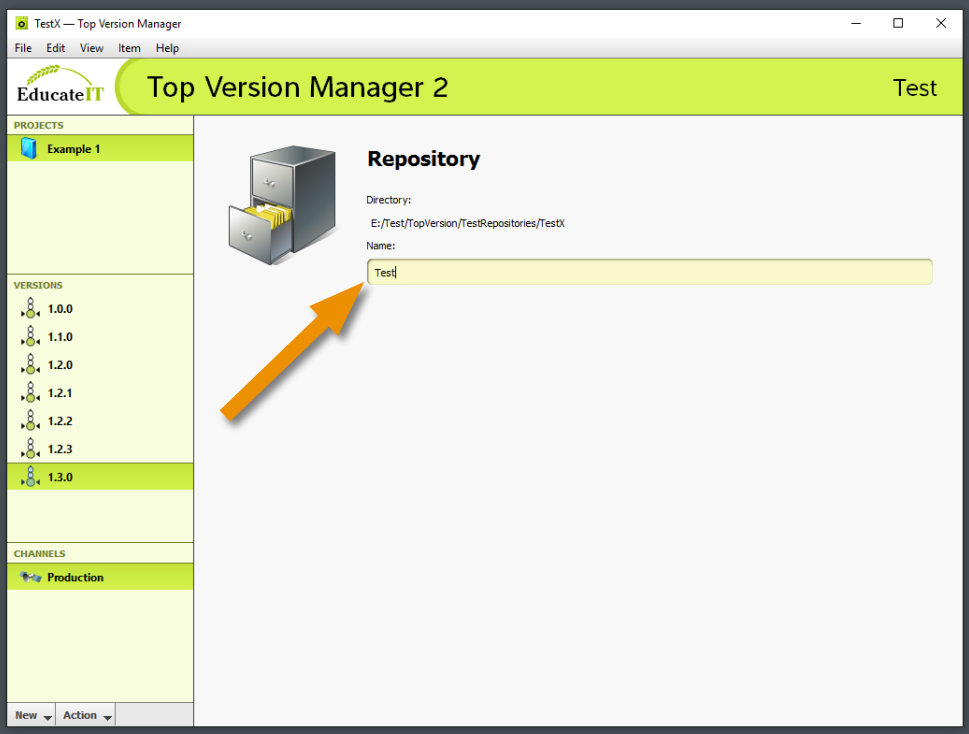
Screen 20: To create reverse delta packages, you have to check this check box.

Add a Repository Name

If you converted a repository from version 1 to the latest version, there is a feature to add a repository name. It is displayed in the top right corner of the repository manager. You can activate this name if you follow these steps:

- Open the repository without name.
- Select in the menu: View and select Show Repository Page.
- On this page, there is a field Name as shown in Screen 21 on page 69.
- Enter a new repository name.

The repository name is stored if you publish/save the repository on disk. To remove a repository name, follow the steps above, but delete the name and save the empty field.



Screen 21: The visual repository name in the top right corner.

Chapter 6: Selector System

This chapter describes the *Top Version Selector* system which consists of the *Selector* service and *Selector* client.

Mode of Operation

The *Top Version Selector* system provides a restricted interface to switch the version of a channel in a repository. The primary goal is, to provide an user interface for support employees in the field to trigger the version switch manually at the right point in time. The secondary goal is to create a privilege separation, where only the service requires write access to the repository and the client does not need any special privileges at all.

System Elements

See Illustration 7 on page 73 for a basic overview of the system. After start, the client is connecting to the service and requests a configured profile. The service checks the profile with the repository and send the profile with all its switch elements back to the client. Next the client presents this profile visually to the user.

If the user makes version changes, this change is sent to the service. Again, the service checks the profile and on success updates the channel version of the repository.

Information Flow

There are a number of parameters which have to be known to start the client. Most parameters can be specified using a configu-

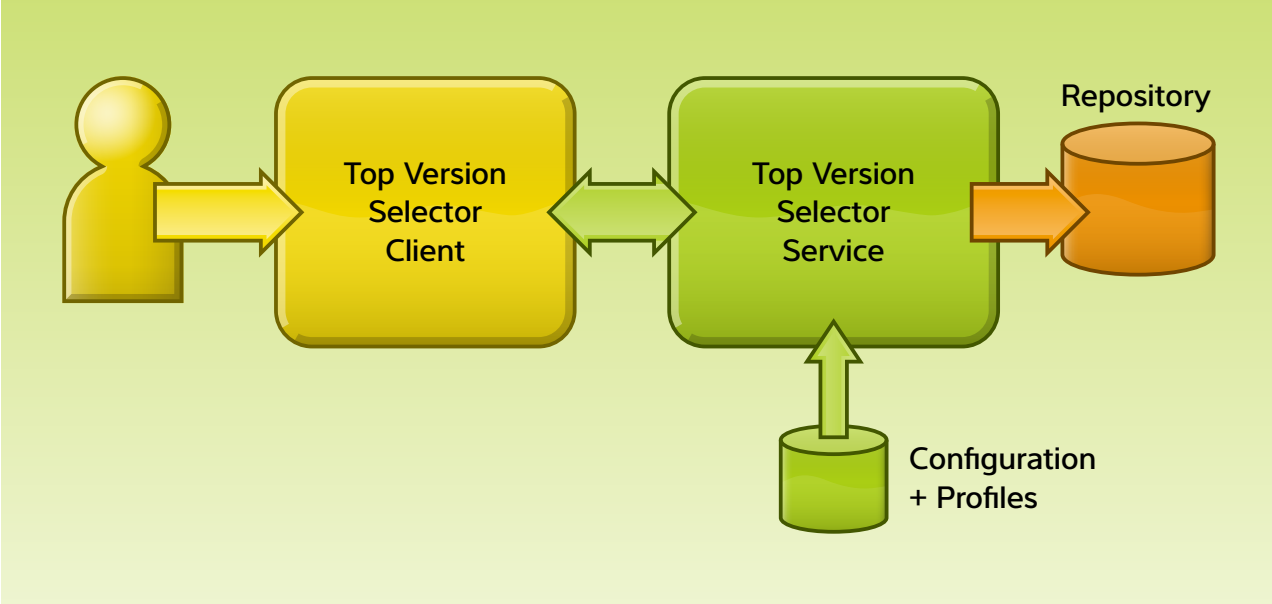


Illustration 7 Overview of the Top Version Selector system using the client.

ration file, but also using command line options. The command line options always have precedence over the configuration file.

Host Name and Port

These parameters are required to connect via TCP/IP to the service. They can be set in the configuration, but you can also overwrite the default values with command line options.

The idea behind the command line options is to select a different service, depending on the repository you would like to change.

While in most situations, one service can change all repositories using different profiles, there are special cases. One case is, if the repositories are stored in different domains and require different permissions to write to them. In this special case, you simple setup multiple *Selector* services - one in each domain - each running under a user which has the required permission for the repository in its domain.

Key and Secret for the Connection

The key and secret for the connection can only be set in the configuration file. In almost all cases this is enough, just adding the same secret pair to all services which has to be accessible by this client.

In a very special case you really need different keys and secrets to connect to different services, have a look into the *Configuration B* system manual. It is possible to create multiple different configuration files and select one using command line options.

Name of the Profile

The name of the profile can optionally set in the configuration file. In this case you do not have to specify any command line options to start the client.

Usually you will specify the profile using a command line option which you add to the link starting the client.

Client Usage

After start, the client will retrieve the profile from the *Top Version Selector* service and present the configured profile as shown in Screen 22 on page 75.

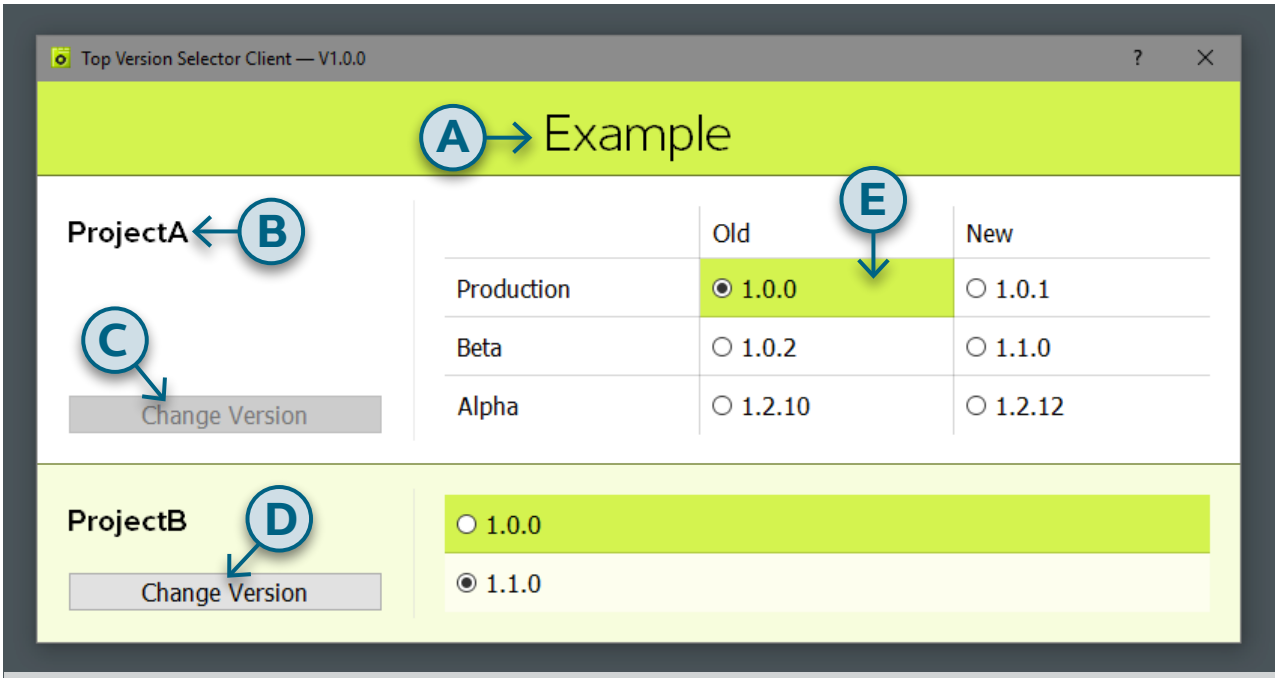
The top area Ⓐ shows the title of the current profile. This title should be unique and meaningful, so a user can easily see if he clicked the correct link if there are multiple profiles configured.

Each project section Ⓑ has a label with the project name. This is important if you configure multiple projects as shown in the example.

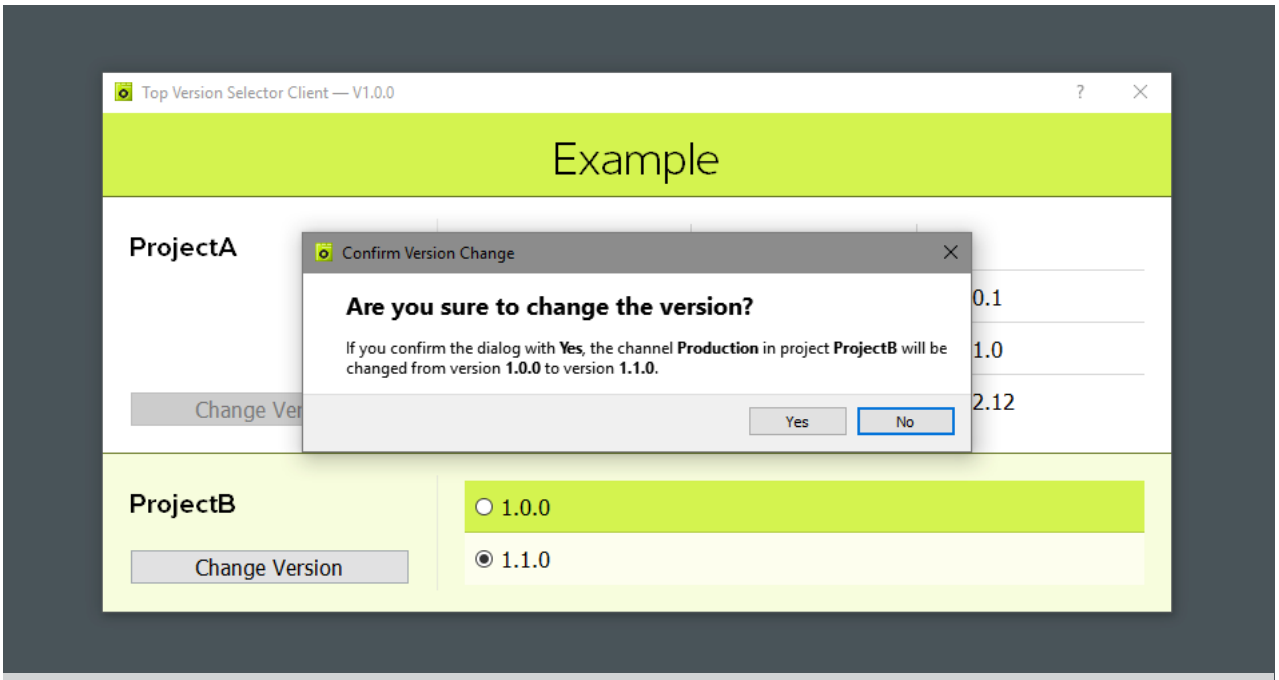
The State of the Change Button

After the start, the Change button is inactive and can not be clicked as shown at label Ⓒ. As soon the user changes the version by clicking one of the switches, the Change button turns active as shown at label Ⓓ.

The current active version is marked with a special background shown at label Ⓔ. This mark makes sure, the current active version is visible even the user already selected a new version using a different switch.



Screen 22: The elements of the client interface.



Screen 23: The confirmation dialogue after clicking the Change button.

Clicking the Change Button

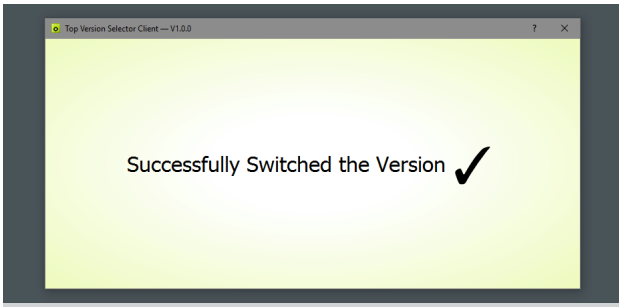
A confirmation dialogue is displayed after clicking the Change button. This is shown in Screen 23 on page 76. This dialogue displays a summary what will happen if the user confirms the action using the Yes button. If the user clicks the No button, no action is executed and he is back in the main dialogue.

After Confirming a Change

After confirming the change, a progress message is displayed, shown in Screen 24 on page 76. Usually the actual change will take only a very shown time, for this reason this progress screen may not be visible at all. After a successful version change, a success message is displayed, shown in Screen 25 on page 76. This message is displayed for three seconds and disappears automatically.

Back in the main dialogue, the changed current version is displayed by moving the current version mark to the new version.

Screen 24: Progress message.



Screen 25: Success message.

Security and Protocol

The *Top Version Selector* system implements a comprehensive number of security features which guarantee the most secure operation in any enterprise network.

Protocol Overview

- The connection is always initiated from the client side to the service.
- One single *TCP/IP* connection is needed to connect from the client to the service.
- The used *TCP/IP* connection works over any number of *NAT* boundaries if it is initiated from the *Selector* client which is inside of the mapped network – on the client side.
- The *TCP/IP* connection can also be made through any kind of proxy, but this has to be transparent for the application. There is no native proxy support in the client.
- Only one single port is used for the connection. This port can be configured to avoid any conflicts with other applications. The port on the client side do not have to match the service side if it gets translated in a router or similar, as long the TCP/IP connection can be safely established.
- Information will flow in both directions over the same connection.
- The data rate is very low and is not time critical.

Authentication

The authentication of the client to the service is a multilayered system, each built of the other one.

SSL Layer

- All products share the same certificate authority which is used to verify the client and service certificate mutually. Each product has a built-in public key of the certification authority which is used for the verification.
- Each product also contains a copy of the public key of any peer it connects to. The public key of the peer is verified on each connection using this local copy of the key.

- Only cyphers which are known as secure are used for the actual *SSL* connection. Any weak ciphers are rejected. Our products usually connect using the `DHE-RSA-AES256-GCM-SHA384` cypher via `TLSv1.2`. There is some room for less secure cyphers, but the security is never lower than `AES128` with `SHA256`. The actual used cypher suite is always visible in the log file.

Application Layer

- If a secure *SSL* connection is established, the client authenticates to the service using a key and secret system. This is similar to a regular *username/password method*, yet the key and secret pairs have to be generated using the *EducateIT Secret Generator*. This generator makes sure, the generated credentials are generated with the correct amount of entropy and does not allow insecure keys and secrets.
- There can be any number of key/secret pairs stored on the service side. On the service side only a *SHA3-512* has with a salt is stored, not the secret itself. The stored value is hashed 32 times to increase the complexity. This should actively prevent retrieving a secret from this hashed value.
- For each key/secret pair, the valid *IP range* for a connection can be limited.
- The available services can be set on a key/secret basis.

Risks and Weaknesses

We take security very serious in our products and always try to implement the best strategies to protect any communication between our systems and any data handled or processed. Nevertheless we *assume* our products are running in a *secure* enterprise environment and not on public accessible, infiltrated or hacked systems.

- Our products have built-in private keys to mutually verify the *SSL* connections. A sophisticated attacker can extract these keys from the software if he gets access to the executable. After extracting these keys and if he is able to setup a *man in the middle attack*, he will be able to read and manipulate the data stream between the client and server.
- The key and secret can be copied from the client configuration and used with any other client to establish a connection to the service.

While the implemented measures to protect the data and communication will be suitable for most companies, do not hesitate to contact us if you have further requirements for additional protection and security mechanism.

Best Practices

- Make sure only administrators and the application itself can read the configuration files and executable.
- Encode any password and secret using the *Password Encoder* tool to make it harder for simple extraction. Note, it still can be copied unmodified from one configuration to another configuration of the same product. The tool just protects the plain text of the password or secret, so it can not be easily extracted.
- If possible, always limit the *IP address range* for a key/secret pair to the minimal acceptable range or a single *IP address*.
- In case our products make a connection over the internet, leaving the secure company network, *always* add an additional layer of security. E.g. make the connection via a secure *VPN tunnel*.
- Limit the access to client applications in a way only employes with the required permissions can start the application.
- Use *firewall rules* to limit the connections to the service and allow only valid sources with the correct protocol.
- Let the service always run with exactly the required privileges for the task.

Client Configuration

The *Top Version Selector* client is using the "*Configuration B*" system for the configuration. See Illustration 8 on page 82 for the structure of the client configuration and Listing 11 on page 82 for an example configuration. The following sections describe each value in detail.

host

The `host` value configures the *IP address* or *host name* where the *Top Version Selector* service is running. See the following box for a number of different ways to configure the `host` value.

```
10.50.109.12
2001:0db8::5785:f2e9
server03
server03.example.com
```

port

The optional `port` value configures the port which is used to connect to the service. If you omit this value, the default port `19680` is used for the connection.

apiKey

The `apiKey` value configures the key used to authenticate to the service. This key and its secret have to be generated using the *EducateIT Secret Generator* application.

apiSecret

The `apiSecret` value configures the secret used to authenticated to the service. This secret and its key have to be generated using the *EducateIT Secret Generator* application.

profile

You can use the `profile` configuration value to set a default profile to retrieve if no command line options are specified. The profile identifier has to match one from the service configuration.

Module	`TopVersionSelectorClient`
Value	`apiKey` — String
Constraints	▪ Must not be empty.
Value	`apiSecret` — Password
Constraints	▪ Must not be empty.
Value	`host` — String
Constraints	▪ Must not be empty.
Value	`port` — Optional — Integer
Constraints	▪ Minimum value: 1 ▪ Maximum value: 65535
Default	`19680`
Value	`profile` — Optional — String
Constraints	▪ Maximum length: 64 character(s). ▪ Has to match the following regular expression: `[-a-zA-Z0-9_]+`

Illustration 8 The structure of the *Top Version Selector Client* configuration.

```xml
<?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="host">127.0.0.1</Value>
    <Value name="apiKey">[key]</Value>
    <Value name="apiSecret">[secret]</Value>
  </Module>

</Configuration>
```

Listing 11 Example configuration for the *Top Version Selector* client.

Client Command Line

The *Top Version Selector* client supports a number of command line options. All command line arguments have precedence over the file based configuration. See Table 4 on page 83 for a list of all supported command line arguments.

See the following example how you start the client using these command line arguments:

```
TopVersionSelectorClient.exe --host=127.0.0.1 --profile=test1
```

Argument	Description
`--host` `-h`	Use this argument to change the host *IP address* or *host name* to connect to. If you specify this parameter, it will have precedence over the host specified in the configuration file.
`--port` `-u`	Use this argument to change the port to connect to. If you specify this parameter, it will have precedence over the port specified in the configuration file.
`--profile` `-p`	Use the `profile` argument to specify a profile to retrieve from the service. If there is already a profile set in the configuration file, this argument will have precedence.

Table 4 Command line arguments for the *Top Version Selector* client.

Service Configuration

The *Top Version Selector* client is using the "*Configuration B*" system for the configuration. See Illustration 9 on page 86 for the structure of the service configuration and Listing 12 on page 90 for an example configuration. The following section describe each value in detail.

The `network` List

The `network` list configures all settings for the different network interfaces of the service. Each entry in the list can configure one single interface. If you omit this list or leave it empty, each interface is configured with default values.

`network » interface`

The required `interface` value chooses the interface which shall be configured by this entry. There are two valid values, explained in the following table.

Interface	Description
`message`	The `message` value configures the settings for the message interface, which is used by the *Selector* client.
`http`	The `http` value configures the *JSON/HTTPS* interface which can be used from custom scripts and extensions.

`network » port`

The optional `port` value configures the port where the selected interface is listening on. If you omit this value, the default port for the specified interface is used. The default ports are:

Interface	Port
Message protocol	19680
JSON/HTTPS protocol	19681

`network » scope`

The `scope` value configures the accepted range of incoming connections. There are three valid values, shown in the following table.

Module	`Application`
List	`network` — Optional
Definition	List Entry Definition
Value	`interface` — String
Constraints	▪ Must not be empty. ▪ Has to be one of the following values: `message`, `http`
Value	`port` — Optional — Integer
Constraints	▪ Minimum value: 0 ▪ Maximum value: 65535
Default	`0`
Value	`scope` — String
Constraints	▪ Has to be one of the following values: `None`, `LocalHost`, `Any`
Default	`LocalHost`
List	`profiles`
Constraints	▪ Must not be empty.
Definition	List Entry Definition
Value	`file` — String
Constraints	▪ Has to be an absolute path to an existing file.
Value	`identifier` — String
Constraints	▪ Must not be empty. ▪ Maximum length: 64 character(s). ▪ Has to match the following regular expression: `[-a-zA-Z0-9_]+`
List	`secrets`
Definition	List Entry Definition
Value	`authorizedNetworks` — Optional — String
Value	`authorizedServices` — String List

Illustration 9 The structure of the *Top Version Selector Service* configuration.

Constraints

- Must not be empty.
- Each element of in this list has to be one of the following values: `Http`, `Message`

Value

`key` String

Constraints

- Must not be empty.

Value

`label` String

Constraints

- Must not be empty.

Value

`secret` Password

Constraints

- Must not be empty.

Scope	Description
`None`	With this setting the interface is not started on the given port, therefore no connections can be accepted.
`LocalHost`	Using this setting, only connections from the same host are accepted. This is useful for testing or if all services run on the same host. It is the default setting.
`Any`	With this setting, connections from the whole network are accepted.

The `secrets` List

The `secrets` List controls which systems can access the APIs of the service. You should generate a own secret pair, using the *EducateIT Secret Generator* for each client which would like to access the statistic server. You should also encode the secret on the client using the *EducateIT Password Encoder*. The secret on the client is like a password and you should protect it like one.

You can further limit the access using the `authorizedNetworks` value. If you know a client is always connecting from the *IP range* `10.0.0.0/24`, specify this network in `authorizedNetworks` to limit the access to this specific key.

`apiSecrets » label`

The `label` value is used to identify an entry of the secrets list. The label helps to easily know which secret was used to access the *API*.

`apiSecrets » key`

The `key` value configures the key for the key/secret pair. Copy this value from the generated secret pair using the *EducateIT Secret Generator*.

`apiSecrets » secret`

The `secret` value configures the secret for the key/secret pair. It is the server secret generated with the *EducateIT Secret Generator*.

`apiSecrets » authorizedNetworks`

The `authorizedNetworks` value can contain a whitespace separated list of networks which are authorized to access the given secret. Only hosts which *IP address* matches one of the configured networks can use the key/secret pair for authentication.

`apiSecrets » authorizedServices`

The `authorizedServices` value configures one or more services which can be accessed using the given key/secret pair. It has to be a whitespace separated list of service names.

Valid service names are shown and explained in the following table.

Service Name	Description
`Message`	The `Message` identifier is for the message protocol which is used by the client to connect to the service.
`Http`	The `Http` identifier is for the *JSON/HTTPS* protocol which can be used with own scripts and applications to access the service.

The `profiles` List

The `profiles` list configures any number of profiles which are provided by the service. Each requires an unique identifier and the path to an XML file with the profile definition.

`profiles » identifier`

The `identifier` value configures an unique identifier which is used to address a configured profile from the client. It has to be less or equal 64 characters long and only contain letters, numbers, the underscore and hyphen character.

`profiles » file`

The `file` value configures the absolute path to the XML file which contains the profile definition. While the path has to be absolute, you can use the features of the *Configuration B* system to create relative paths to the application or to the configuration file.

Statistic Server Support

The *Top Version Selector* service supports the statistic system. You can enable the reporting by adding the `StatisticSystem` module to the configuration as described in the manual "*Statistic System – Administrator Manual*". Please read this documentation for all details how to configure the reporting.

Example Configuration

Listing 12 on page 90 is an example configuration for the service. It configures the message interface to receive connections from the local host, which is suitable for testing on the same machine. The *JSON/HTTPS* interface is disabled using `None` in the `scope` value.

There are two key/secret pairs configured for a client and an Unix script. The actual key and secret is replaced with the placeholder `[key]` and `[secret]`, you have to use real values there to make the configuration work.

In the `profiles` list are two entries for profile definitions which are in the same directory as the configuration file. This is made possible using the `thisDirectory` placeholder from the *Configuration B* system. Read the manual "*Configuration B – Administrator Manual*" for all details about the different possible placeholders in the configuration.

The last part of the file is the log configuration using the `Log` module. In this example, the simplest configuration is used, just setting a path to the log file. Read the manual "*Log System – Administrator Manual*" for all details how to setup the log system for your own needs.

```xml
<?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">

    <List name="network">
      <ListEntry>
        <Value name="interface">message</Value>
        <Value name="scope">LocalHost</Value>
      </ListEntry>
      <ListEntry>
        <Value name="interface">http</Value>
        <Value name="scope">None</Value>
      </ListEntry>
    </List>

    <List name="secrets">
      <ListEntry>
        <Value name="label">Client</Value>
        <Value name="key">[key]</Value>
        <Value name="secret">[secret]</Value>
        <Value name="authorizedServices">Message</Value>
      </ListEntry>
      <ListEntry>
        <Value name="label">UnixScript</Value>
        <Value name="key">[key]</Value>
        <Value name="secret">[secret]</Value>
        <Value name="authorizedServices">Http</Value>
      </ListEntry>
    </List>

    <List name="profiles">
      <ListEntry>
        <Value name="identifier">profile1</Value>
        <Value name="file">●thisDirectory●/profile1.xml</Value>
      </ListEntry>
      <ListEntry>
        <Value name="identifier">profile2</Value>
        <Value name="file">●thisDirectory●/profile2.xml</Value>
      </ListEntry>
    </List>

  </Module>

  <Module name="Log">
    <List name="FileWriter">
      <ListEntry>
        <Value name="LogFilePath">c:\TopVersion\SelectorService.log</Value>
      </ListEntry>
    </List>
  </Module>

</Configuration>
```

Listing 12 Example configuration for the *Top Version Selector* service.

The Profile Definition

Each profile is defined in a XML document. See Illustration 10 on page 92 for the basic structure of this XML document and Listing 13 on page 94 for a complete example. There is also a schema which verifies the profile document, shown in Listing 18 on page 102.

The `Profile` Element

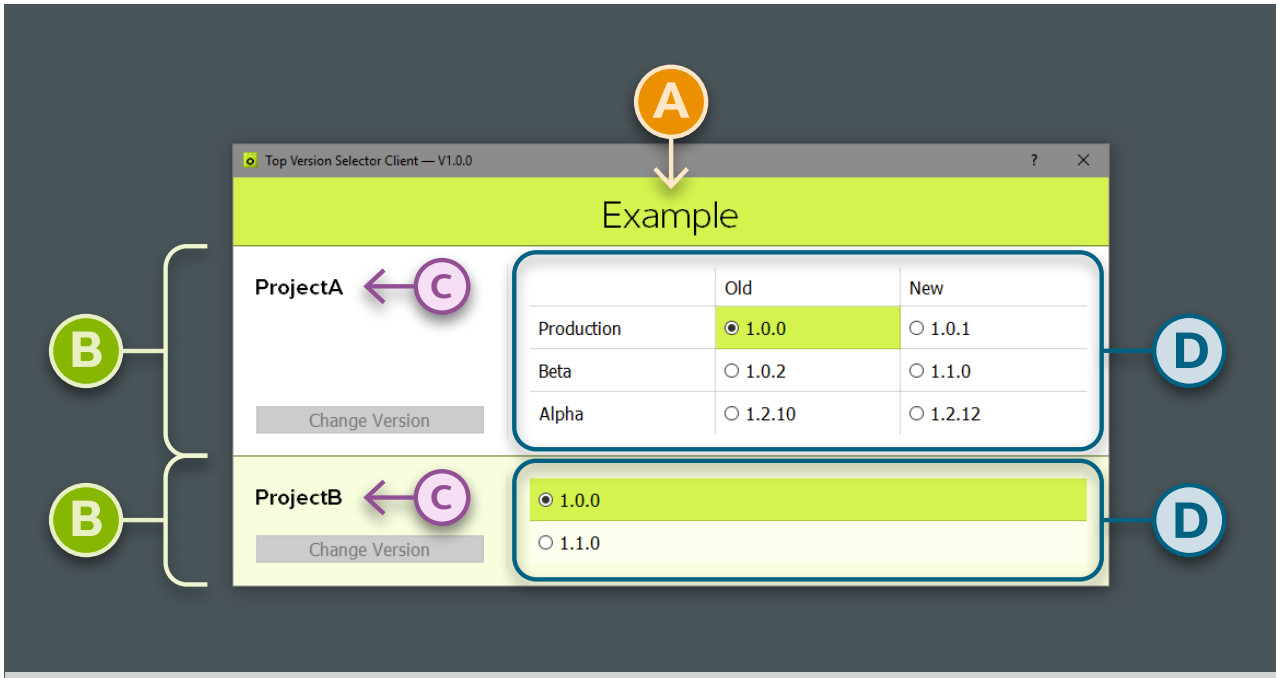
The profile definition starts with the document element, the `SelectorProfile` tag. This tag requires the attribute `version` which has to be set to 1 for the current version of the system.

The `Title` Element

Inside of the document element, there is the `Title` element, which contains the title shown in the client. See label Ⓐ in Screen 26 on page 91: This is the place where the title is displayed in the client. A title is required for each profile.

The `RepositoryPath` Element

Next element is the `RepositortPath` element. It contains the absolute path to the repository which is updated using this pro-



Screen 26: The profile elements shown in the client.

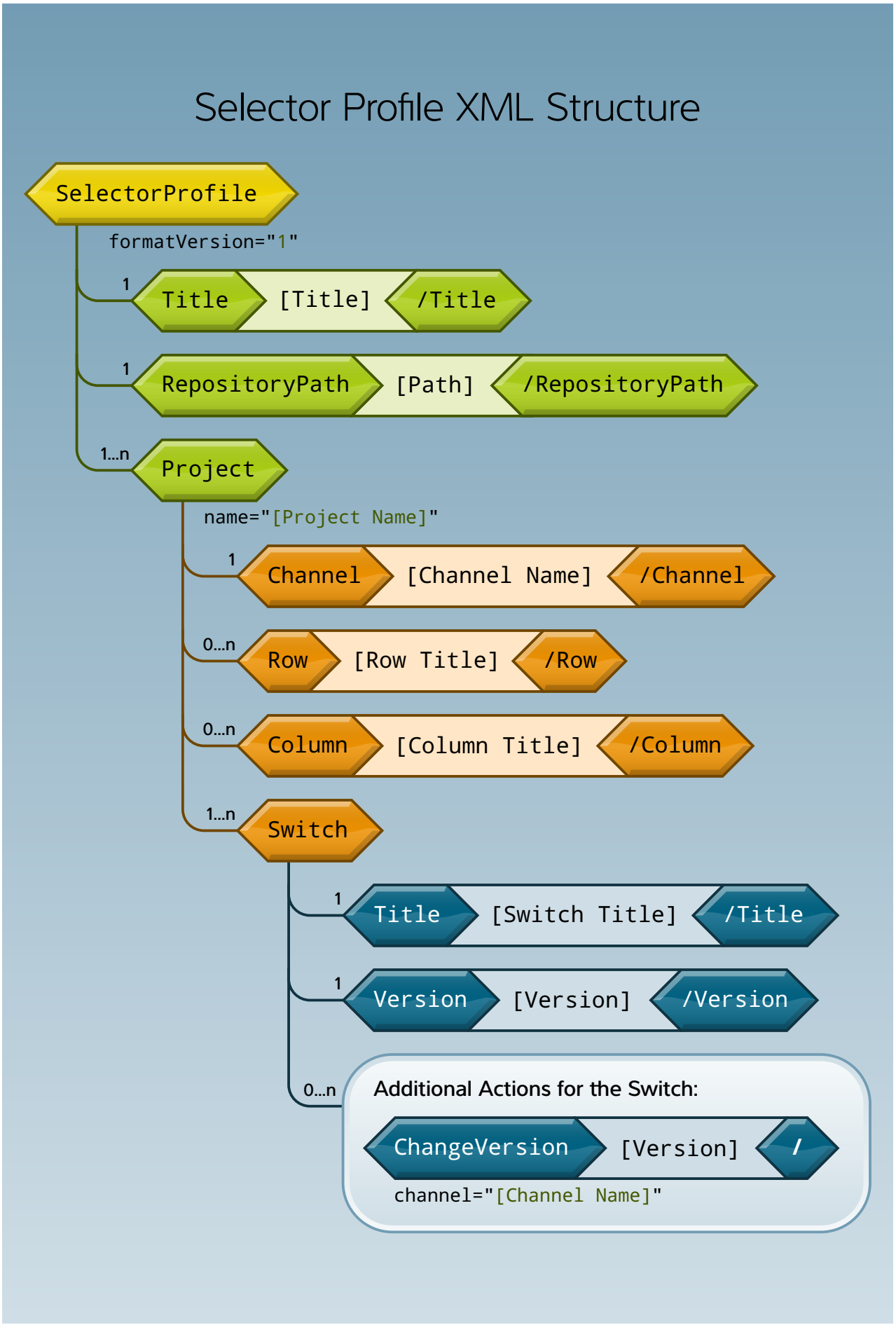


Illustration 10 The structure of the *Top Version Selector* profile XML document.

file. The path has to point to the *directory* of the repository, not to the repository file itself. It must not end in a slash character.

The `Project` Elements

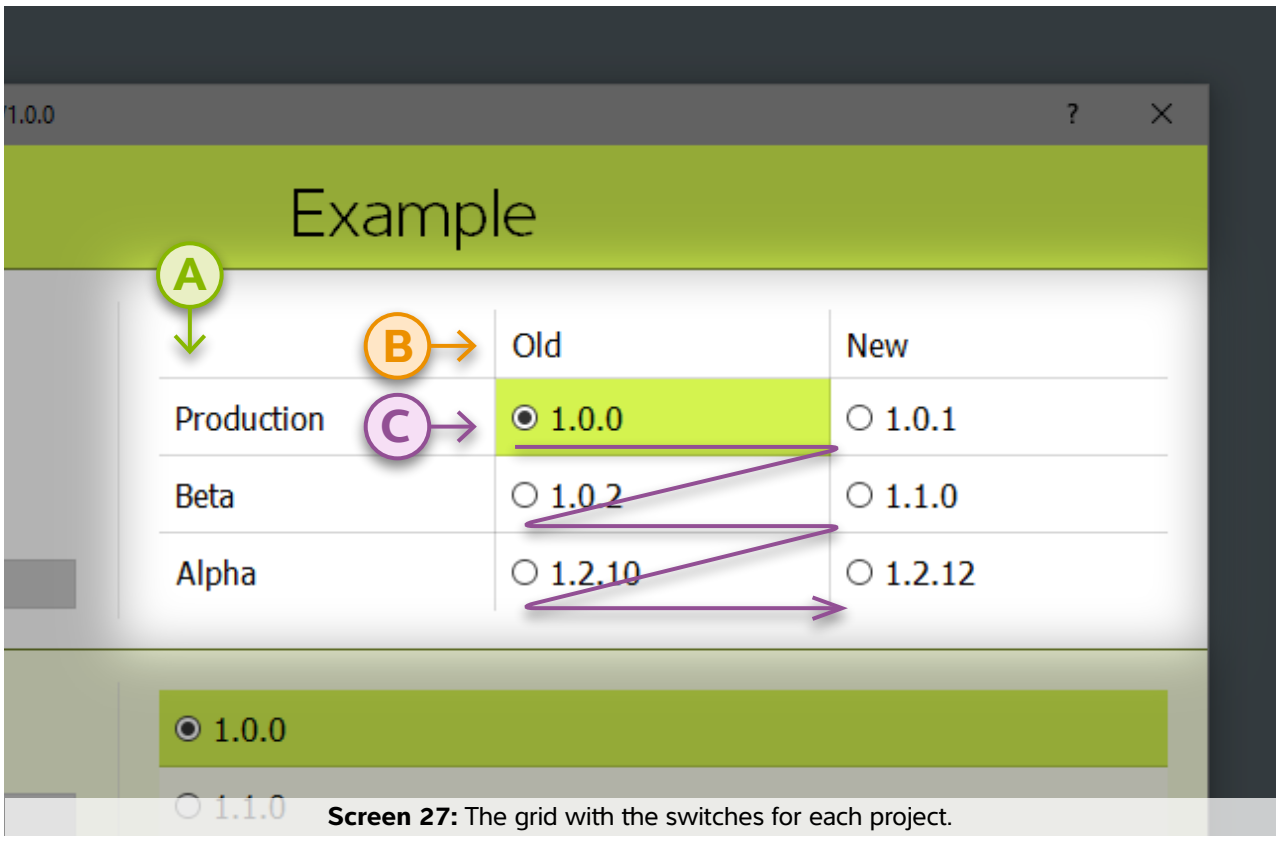
After this path, any number of project definitions can follow. Each enclosed in a `Project` element. The project element requires the attribute `name` with the exact name of the project as value. You find this name in the `repository.xml` file of the repository. Usually you only define one single project per profile. Multiple project definitions only make sense if the projects are related, and the version switch has to be done at the same time.

See label Ⓑ in Screen 26 on page 91: There is a own section in the client for each `Project` element in the profile. The project name is shown in the top left corner Ⓒ of the section. Each section also has its own Change button.

The last element is the grid with all switches, marked with label Ⓓ in Screen 26 on page 91.

The Project Element Structure

The first element in each project has to be the `Channel` element. It selects the primary channel which is modified using the switches. This channel is also used to show the current selection



Screen 27: The grid with the switches for each project.

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

  <Title>Example</Title>
  <RepositoryPath>E:\Test\TopVersion\TestRepositories\TestB</RepositoryPath>

  <Project name="ProjectA">
    <Channel>Production</Channel>
    <Row>Production</Row>
    <Row>Beta</Row>
    <Row>Alpha</Row>
    <Column>Old</Column>
    <Column>New</Column>
    <Switch>
      <Title>1.0.0</Title>
      <Version>1.0.0</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
    <Switch>
      <Title>1.0.1</Title>
      <Version>1.0.1</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
    <Switch>
      <Title>1.0.2</Title>
      <Version>1.0.2</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
    <Switch>
      <Title>1.1.0</Title>
      <Version>1.1.0</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
    <Switch>
      <Title>1.2.10</Title>
      <Version>1.2.10</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
    <Switch>
      <Title>1.2.12</Title>
      <Version>1.2.12</Version>
      <ChangeVersion channel="Beta">1.0.0</ChangeVersion>
    </Switch>
  </Project>

  <Project name="ProjectB">
    <Channel>Production</Channel>
    <Switch>
      <Title>1.0.0</Title>
      <Version>1.0.0</Version>
    </Switch>
    <Switch>
      <Title>1.1.0</Title>
      <Version>1.1.0</Version>
    </Switch>
  </Project>

</SelectorProfile>
```

Listing 13 Profile definition example with two project entries.

version. Make sure the channel name uses the exact spelling as defined in the `repository.xml` document.

Row and Column Titles

Next you can define any number of row titles using `Row` elements. They are displayed as shown with label Ⓐ in Screen 27 on page 93. After the row labels, you can define any number of column titles using `Column` elements. See label Ⓑ for these. Row and column titles are completely optional and only recommended if you have a large number of logically grouped switches. If you omit these elements, there will be no visible header rows with the titles. You can also just specify row titles, but no column titles – or just column titles and no row titles.

While there is technically no limit to the number of rows and columns, you should keep things as simple as possible. If you have more than three rows or columns, you probably misunderstand the idea behind the *Selector* system.

The Switches

After the `Channel` and optional `Row` and `Column` elements, any number of `Switch` elements follow. You must at least specify one switch element, but obviously two or more elements are needed to provide any choice to the user.

The switches are displayed in the order you define them in the document. See label Ⓒ in Screen 27 on page 93: The switches are added from left to right, from top to bottom, filling the grid. If there are no titles defined, they are just added from top to bottom in one single column.

If there are more switches than defined row titles, empty row titles are automatically added at the end of the layout.

The `Title` Element

Each switch need a `Title` element. This title is displayed in the client. The title can be different from the version, but make sure you update it with every version change to avoid any confusion.

You can set the title for example to “Old” or “New”, or even to “Production” or “Test” instead of providing the exact version number. This can make things simpler for the user and more consistent if the underlying versions change.

The `Version` Element

The `Version` element is the second required element of each switch. It takes the exact version name as it is defined in the `info.xml` file for each project. This version is used to actually

change the channel version, but it is also used to determine which is the currently selected switch.

Additional Actions

Each switch can add any number of additional actions which are executed if the user changes the version of the channel. Currently there is only the `ChangeVersion` element, which changes the version of a different channel as well.

You can use the `ChangeVersion` element if for example the test channel should be changed to the next beta version of the software, as soon the production channel is switched to the new version.

There is no limit to the additional actions, but be careful with additional version changes. They are not visible to the user and it is important he knows about this additional changes.

Profile Verification

Each profile is verified at various points in time. The first verification is done if the *Top Version Selector* service starts. After this check, the profile is verified each time when the user starts the client and requests the profile from the service. A last check is done just before the version of a channel is changed.

If there are any problem while the service is started, the service will cancel the start and report the problem in the log file. Once the profile was verified at the start and the client requests a profile, ad additional check is made. If there is any problem, the client will not start and display a message with the problem. A more detailed error description is written in the log file on the service side.

Perfomed Checks

- Checks the repository path and if the repository file can be written by the service.
- Checks if the configured project and channel exists.
- Checks if each configured version exists in the project.
- Checks if the switch titles and versions are unique.
- Checks if all additional actions are valid.

Chapter 7: Appendix

This appendix contains the *XML schema* for the different *XML* files in the repository and for the *Selector* service. You should got this schema files along with the software to verify the *XML* documents, if you make manual changes or using own software to modify the repository.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/Repository/2"
  xmlns="http://educateit.ch/software/TopVersion/Repository/2"
  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="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:attribute name="visibleName" type="xs:string" use="optional" />
  </xs:complexType>

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

Listing 14 Schema for the `repository.xml` file.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/Project/2"
  xmlns="http://educateit.ch/software/TopVersion/Project/2"
  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 15 Schema for the project info file `info.xml`.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/ProjectVersion/2"
  xmlns="http://educateit.ch/software/TopVersion/ProjectVersion/2"
  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 16 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/2"
  xmlns="http://educateit.ch/software/TopVersion/ProjectStatus/2"
  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 17 Schema for the installation status file.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://educateit.ch/software/TopVersion/SelectorProfile/1"
  xmlns="http://educateit.ch/software/TopVersion/SelectorProfile/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:simpleType name="ShortText">
    <xs:restriction base="xs:string">
      <xs:maxLength value="64" />
      <xs:minLength value="1" />
      <xs:pattern value="^[^\p{Zl}\p{Zp}\p{C}]+$" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="ChangeVersion">
    <xs:simpleContent>
      <xs:extension base="VersionName">
        <xs:attribute name="channel" type="EntryName" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="Switch">
    <xs:sequence>
      <xs:element name="Title" type="ShortText" />
      <xs:element name="Version" type="VersionName" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="ChangeVersion"
type="ChangeVersion" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="Project">
    <xs:sequence>
      <xs:element minOccurs="1" maxOccurs="1" name="Channel" type="EntryName" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Row" type="ShortText" />
      <xs:element minOccurs="0" maxOccurs="unbounded" name="Column" type="ShortText" />
      <xs:element minOccurs="2" maxOccurs="unbounded" name="Switch" type="Switch" />
    </xs:sequence>
    <xs:attribute name="name" type="EntryName" use="required" />
  </xs:complexType>

  <xs:element name="SelectorProfile">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Title" type="ShortText" />
        <xs:element name="RepositoryPath" type="xs:string" />
        <xs:element minOccurs="1" maxOccurs="unbounded" name="Project" type="Project" />
      </xs:sequence>
```

Listing 18 Schema for the *Top Version Selector* profile definition.

```
      <xs:attribute name="formatVersion" type="xs:unsignedByte" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

EducateIT GmbH
Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland
Phone: +41 32 355 39 00
E-Mail: info@educateit.ch
Web: http://www.educateit.ch/