



Configuration B

Administrator Manual

Software Version 1.8.1



EducateIT GmbH

Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland

Phone: +41 32 355 39 00

E-Mail: info@educateit.ch

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

Configuration B – Administrator Manual

Copyright: ©2014 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: 2014-09-09 11:13

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.

Table of Contents

Preface	5
Features.....	5
How to Find the Version Number	6
Files and Locations.....	8
Search Paths.....	8
Product Name	8
File Names	9
Configuration File Structure.....	11
The Configuration Elements.....	12
The IncludeFile tag.....	12
The IncludeDirectory tag.....	13
The Module tag.....	14
The Value Element.....	15
The Types of the Value Element	16
The Group Element.....	18
The List Element.....	18
Placeholders.....	21
Application Directory Placeholder.....	21
This Directory Placeholder	21
Date/Time Placeholder	22
Date/Time Format	22
Computer and Domain Name.....	24
Use Environment Variables	24
Log-On User Placeholders.....	24
Command Line Arguments	25
Long Command Line Arguments.....	25

Short Command Line Arguments	25
The Argument --configFile	26
The Argument --configDirectory	26
Password Encoding	27
Step 1: Locate the Public Key.....	29
Step 2: Encode the Password.....	29
Step 3: Paste It Into The Configuration.....	29
How Secure are Encoded Passwords?.....	30
Security Recommendations	30
Configuration Visualization.....	31
Basic Division	31
Derived Host Configuration	35
The HostGroups Block.....	35
How Hosts are Assigned to Groups.....	36
Derived Configuration Blocks.....	36
Example	37
Migration From Base System.....	39
Automate Conversion With Perl	40
Configuration File Schema	41

Preface

The *Configuration B* system is included in many different EducateIT products. It provides the whole configuration system, including the parsing of command line options. This system is the successor of the *Base Configuration* system, which is used in older products.

For you as user, this document describes all the generic features, which can get used in all our products which are using this configuration system. You probably got this document along with the manual of the software because all generic features are described here, while the software manual just contains the documentation of the specific values for the product.

Be aware that there are probably multiple versions of this configuration system. Later versions can include features which are not supported in the version described in this document. Make sure that you check the version number which is used in the software, and compare it to make sure you are reading the correct document. See section [“How to Find the Version Number”](#) on page 6 how to find the correct version number for this component.

Features

XML Based Format

The configuration is based on a XML format, which is well standardized. Beside of the clear format, also the encoding of the file is automatically defined in the header, which makes this format a good choice for configuration.

See [“Configuration File Structure”](#) on page 11 for details.

Simple Syntax Validation with Schema

Because the structure of the configuration files is based on a common schema, you can simple verify the syntax of your configuration file in any XML editor or with command line tools. Usually the schema for the configuration file is installed in the application directory with the name `ConfigurationB.xsd`. If this is not done, please request the schema from us.

See [“Configuration File Schema”](#) on page 41 for a copy of the schema.

Encoded Passwords

To prevent not authorized people to read passwords from configuration files it is possible to encrypt and encode them for the configuration.

See [“Password Encoding” on page 27](#) for details.

Configuration Modules

For huge applications, each part is configured in a own section of the configuration. This way you can reuse existing configurations for certain modules, if you worked with an application which used the same component before.

See [“The Module tag” on page 14](#) for details.

Spread Configuration Across Multiple Files

You can include other configuration files from your base configuration and split the configuration into multiple files. Or you can just include the configuration from a network share to easily update it without the need of copy it to all clients.

See [“The IncludeFile tag” on page 12](#), [“The IncludeDirectory tag” on page 13](#), [“The Argument --configFile” on page 26](#) and [“The Argument --configDirectory” on page 26](#) for details.

Placeholders for Paths, Dates and More

In each value you can use placeholders which get replaced at the time the configuration is read. There are placeholders for the application path or the directory where the configuration file is read from, and there is a placeholder to insert dates in various formats and offsets. They can used to automatically build paths and file-names for configuration.

See [“Placeholders” on page 21](#) for details.

Derived Modules for Host Groups

For some multitier applications it is possible to create derived module configuration on based on the IP address of the host which is requesting the configuration from the server.

See [“Derived Host Configuration” on page 35](#) for details, and check the manual of the application if this feature is enabled.

How to Find the Version Number

If you got this manual along with the software product, there is no need to verify the version number of the configuration system. If you are configuring an software, and search for the right manual, then follow the next steps to locate the version number.

The version number of the configuration system can be found on the detail page of the *about window*. To open the about window you probably have a button called “About...” on the user interface. Or you find a menu called “Help”, and there is a submenu called “About...” which opens this window. If neither of these is available, try double click on the bar which contains the application title.

In the about window you will find a small circle showing the letter “i” in the bottom right corner. Click on this circle to display the detail page. See “[The about window.](#)” on [page 7](#) for an example how the *about screen* looks like.

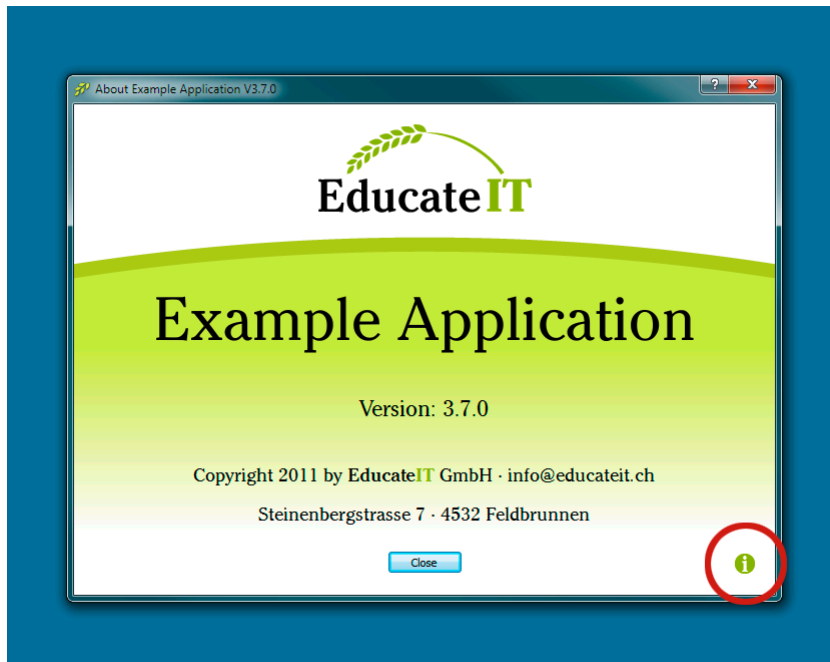


Illustration 1 The about window.

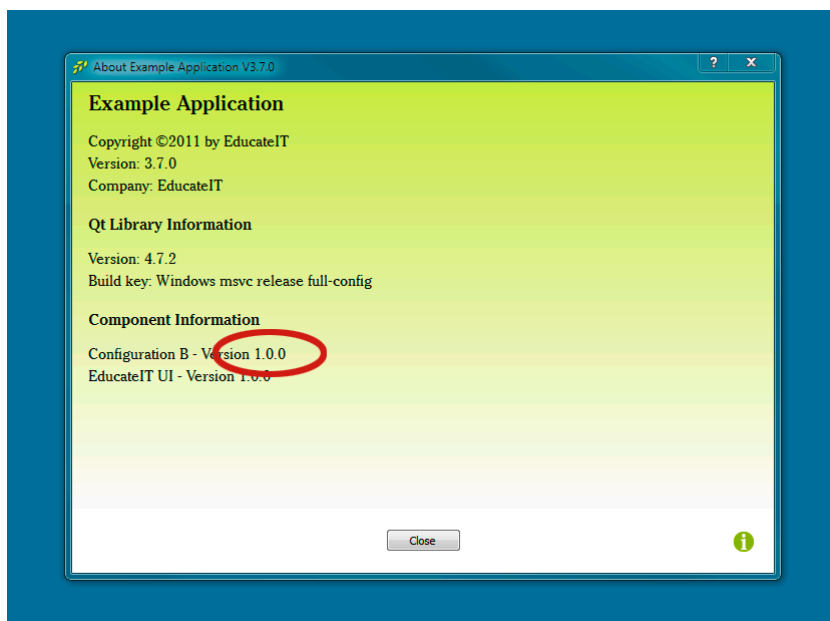


Illustration 2 The detail page of the about window.

The detail pages summarize various information about the internal structure of the application. Search for a component section, and search in the list for an entry which is called “Configuration B”. The number on the right of it is the version number of the configuration system. See [Illustration 2](#) on [page 7](#) where to look for the component version number.

Compare this version number with the version number on the front page of this manual. If there is a difference, in the first or second number, probably some features have changed. If the version number of the configuration is newer than this manual, request a newer version of this manual from our company. If the version number is lower, check the appendix of this document for the changes.

Files and Locations

There is a default behaviour where the system is searching for configuration files. But please note, this default behaviour can get changed or extended by the application itself. A changed behaviour should be documented in the manual of the software product. This document describes only the default behaviour which should be true for most products.

Search Paths

There are some built-in search paths which are scanned for configuration files. The built-in paths are the following two in the shown order:

- OS Appl. Configuration /EducateIT/ Product Name
- Application Directory (only on Windows)

The location of the OS Appl. Configuration depends on the operating system, and the version of the operating system you are using. In the Table 1 on page 9 there are some examples of operating systems and where this path is located. Please keep in mind, that a user usually can change the location of this path on most Windows operating system versions.

Product Name

The product name which is used in the search paths is exactly the name shown in the about window of the application. This includes the spaces and uppercase letters.

If the product is called “Example Application”, and is running on a computer with Windows 7, the searched path is:

C:\ProgramData\EducateIT\Example Applicaion

File Names

By default, the configuration file has to use the following name:

```
configuration.xml
```

If a file with this name is found in one of the searched directories, it is assumed this is a part of the application configuration.

You can also use the suffix `.conf_b.xml` for configuration files. If a file with this suffix is found in one of the search paths it is added to the list of configuration files. See the following examples of valid configuration file names:

```
Application.conf_b.xml  
part1.conf_b.xml  
module_core.conf_b.xml
```

All found configuration files are read and parsed. After the parsing step all values are verified and assigned.

OS	Version	Path
Mac	OS X	/Library/Application Support/EducateIT/ Product Name
Linux	All	The search paths depends on the installation location of the application. If the application is installed in /usr/bin or /usr/sbin, the path /usr/etc and /etc is searched. But if the application is located in /opt/bin or /opt/sbin only /opt/etc is searched. In the case of an installation in /usr/local/bin, only /usr/local/etc is searched. /usr/local/etc/EducateIT/ Product Name /opt/etc/EducateIT/ Product Name /usr/etc/EducateIT/ Product Name /etc/EducateIT/ Product Name
Windows	97 XP 2000	C:\Documents and Settings\All Users\Application Data\EducateIT\ Product Name %ALLUSERSPROFILE%\Application Data\EducateIT\ Product Name
	7	C:\ProgramData\EducateIT\ Product Name %SystemDrive%\ProgramData\EducateIT\ Product Name

Table 1 The different paths for the configuration on various operating systems.

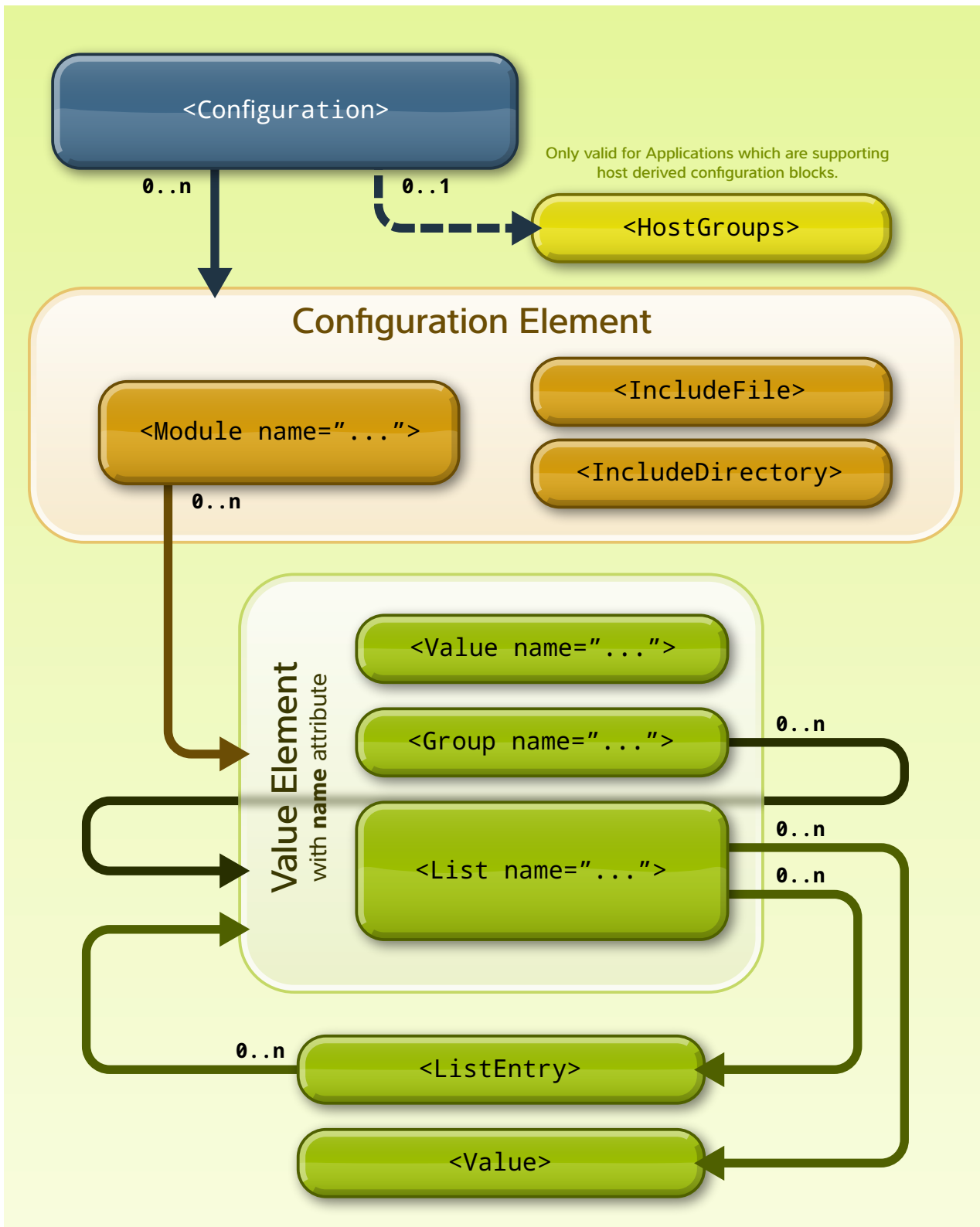


Illustration 3 Structure of the configuration file.

Configuration File Structure

Each configuration file is a valid XML document. You are free to use your preferred encoding for the document, but I strongly recommend the UTF-8 encoding.

Listing 1 on page 11 is showing the document structure of a configuration file. There is the document tag `Configuration` which has one required attribute `version` which has to match the major version of the configuration system. Currently this number has to be set to 1.

The two other attributes define the URI for the schema, and the declaration of the `xsi` namespace. While the first is necessary to verify your document, the second is useful if you like to point to the schema file with an attribute like `xsi:schemaLocation`.

Have a look at Illustration 3 on page 10. It explains the relationships between the XML tags in the document. The blue

```
<?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">

  . . .

</Configuration>
```

Listing 1 The document structure of a configuration file.

box, labelled `<Configuration>`, represents the same tag as shown in Listing 1 on page 11. You see the line with the arrow attached to this box, it is showing the tags which can be contained in this element. The black number range beside the line explains how many of this element can be contained inside of `<Configuration>`. In this case, `0..n` indicates that any number of elements (from zero to any number) are possible.

Some multi-tier applications make use of host derived configuration blocks. They require one single `HostGroups` element before any other configuration element which defines all groups used with the later module blocks. Details about this additional configuration possibilities are described in section “Derived Host Configuration” on page 35.

The Configuration Elements

Inside of the Configuration tag, the Module and the IncludeFile or IncludeDirectory element are allowed. In any number and combination. The Module tag provides *configuration values* for a given *configuration module*. It requires the attribute name, which has to be set to the name of the module which is configured. Usually an application requires one module which is called Application. Be aware, that this name is case sensitive.

Listing 2 on page 12 shows a configuration file with two module definitions and an include tag which reads additional configuration from a second file. In many cases, the configuration file just contains the module definition for the Application module.

The gray elide points (“...”) are showing parts which were elided for readability. This parts are already explained before, or are explained in one of the next sections.

The IncludeFile tag

With this tag, you can include additional configuration files from other locations. It requires as content an absolute path or relative path to the file which you like to include. If you use a relative path, this path is relative to the directory from which the configuration file is read.

This file has to be in the same format and can contain all possible entries without limitation. You can even split a module configuration into two files. You just have to make sure that you do not define the same value in both module blocks.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration version="1"...>

  <Module name="Application">
    ...
  </Module>

  <Module name="LogSystem">
    ...
  </Module>

  <IncludeFile>X:/Example/example.xml</IncludeFile>

</Configuration>
```

Listing 2 A configuration file with some module definitions and an include tag.

If a file from a `IncludeFile` tag is not found, an error is generated and the application cannot start.

There is also a command line argument which you can use to include additional configuration files. See section “[Command Line Arguments](#)” on page 25 for details.

The IncludeDirectory tag

With this tag you can add an additional directory which is scanned for configuration files. The configuration files need to have the suffix `.conf_b.xml`. Each found configuration file with this suffix is loaded and parsed like the primary configuration file.

You can either specify a absolute or a relative directory path. A relative directory path is always relative to the configuration file in which the `IncludeDirectory` tag is placed.

See [Illustration 4](#) on page 13 for an example using a relative path. All files with the correct suffix in the directory are read and parsed.

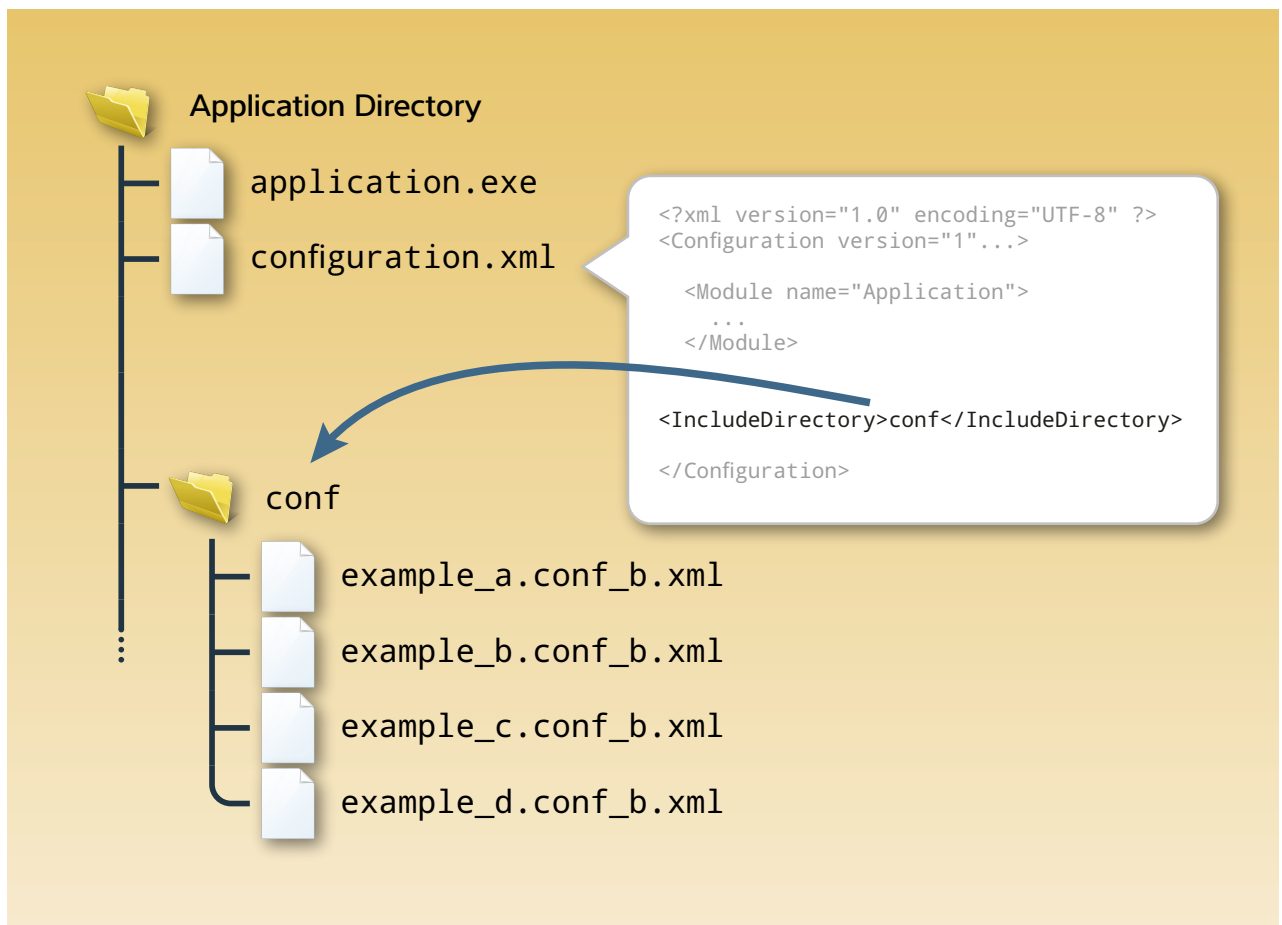


Illustration 4 How the `IncludeDirectory` tag with a relative directory works.

There is also a command line argument which you can use to include additional directories which are scanned for configuration files. See “[Command Line Arguments](#)” on page 25 for details.

The Module tag

The configuration itself is specified inside of the Module tags. Beside of simple values, it can contain groups and lists. Have a look at [Illustration 3 on page 10](#) to see how Configuration, Module, Value, List and Group elements are related to each other.

Each module block refers to a certain configuration module inside of an application, which is addressed with the name attribute. This name is case sensitive and it is defined by the application itself. The manual of the software product should contain the exact module names which have to be configured. Usually there is at least one module called “Application”, which has to be configured.

```
<Module name="Application">

  <Value name="path">•thisDirectory•</Value>

  <Group name="log">
    <Value name="target">c:\log\application.log</Value>
    <Value name="format">PlainText</Value>
  </Group>

  <List name="groups">
    <ListEntry>
      <Value name="id">group1</Value>
      <Value name="title">The First Group</Value>
    </ListEntry>
    <ListEntry>
      <Value name="id">group2</Value>
      <Value name="title">The Second Group</Value>
    </ListEntry>
  </List>

  <List name="files">
    <Value>file1.txt</Value>
    <Value>file2.txt</Value>
    <Value>file3.txt</Value>
    <Value>file4.txt</Value>
  </List>

</Module>
```

Listing 3 Example module block with all four different tag entries.

A module can be configured in one Module block or you can split it over multiple blocks. This makes sense if you like to provide some values from a second configuration file. In this case you just create a second module block with the same name.

Which values you have to put into the configuration block is defined by the application. There are optional and required values. But at any time you can configure each named value only once.

Listing 3 on page 14 shows a fictive example of a module block. It contains all four possible value entries: Value, Group, a List with ListEntry blocks and a List with Value tags. They are explained in detail in the following sections.

The Value Element

The Value element actually contains some configuration value. It contains text, which can have some constraints on it. A configuration value can require an integer for example, so you have to provide a valid integer for this configuration value.

This constraints can not be checked by the XML schema. It does only check the principally syntax of the configuration file. The contents of the Value element are checked at the application start, when the configuration is read and verified.

The manual of the application contains detailed information about each value, about the basic type of the value and additional constraints.

The placeholder Attribute

Each value element can have an optional placeholder attribute. This attribute controls if placeholders in the specified text for the value are replaced or ignored.

The attribute can have either the value Replace or Ignore. Using the latter any placeholder text is ignored and passed without modification to the application.

In the following example the configuration values are parsed on a computer named "EXAMPLE". By default and with the attribute set to Replace the placeholder is replaced. Setting the placeholder to the value Ignore passed the text without modification to the application.

<code><Value>•computerName•<Value></code>	<code>⇒ EXAMPLE</code>
<code><Value placeholder="Replace">•computerName•<Value></code>	<code>⇒ EXAMPLE</code>
<code><Value placeholder="Ignore">•computerName•<Value></code>	<code>⇒ •computerName•</code>

The Types of the Value Element

String Type (Text)

The *string* type can be any kind of text. It can contain any sort of Unicode characters. Be aware that you need to encode XML special characters like `&`, `<` and `>` as `&`, `<` and `>` to keep the XML document valid. Or use the `<![CDATA[ and ]]>` “brackets” to enclose text with many special characters like program code or embedded HTML code.

Of course there can be any kind of constraints by the application, which can limit the characters which can be used or the minimum and maximum length of the text.

```
<Value name="firstValue">Example</Value>
<Value name="secondValue">A longer example text</Value>
```

If you need to pass large portions of text or even HTML or even scripts, you should enclose the value into `<![CDATA[ and ]]>` “brackets” as shown in the next example:

```
<Value name="htmlCode"><![CDATA[
  <p>Some formatted <i>HTML</i> code.</p>
]]></Value>

<Value name="script"><![CDATA[
  if (x > 27 && y < 12) {
    result = "Success!";
  }
]]></Value>
```

Password Type

The password type is a special variant of the “String” or text type. It is specialized on passwords. Passwords are never exposed in log files or shown in error messages.

If the application is supporting password encoding you should always encode the passwords for the configuration using the “EducatelT Password Encoder” tool. See section “[Password Encoding](#)” on [page 27](#) for details.

String List Type

A string list is in fact just a list of *whitespace* separated tokens. See the following example:

```
<Value name="a">apple orange banana</Value>
```

The example defines a list of three values. All of them are separated by whitespace. Whitespace can be any number of spaces, tabs or line breaks. For this reason, the list in the example above can be formatted as shown in the following example with the exact same result.

```
<Value name="a">
  apple
  orange
  banana
</Value>
```

Integer Type

If a value is defined as an integer type, it can contain any positive or negative integer value. But for example no decimal point or fractional digits. There can be other application specific limits, like a minimum or maximum value.

The following example shows some integer values:

```
<Value name="a">-12</Value>
<Value name="b">7000</Value>
```

It is also possible to specify positive integer numbers in a hexadecimal form, just by prefix the hexadecimal value with “0x”. This is shown in the next example.

```
<Value name="a">0xE</Value>
<Value name="b">0x01b9</Value>
```

To configure byte amounts you can also use the suffixes like “kb”, “mb” and “gb”. This suffixes are using a factor of 1000, 1000² and 1000³. This is shown in the following example:

```
<Value name="a">128kb</Value>
<Value name="b">20 mb</Value>
<Value name="c">177 GB</Value>
<Value name="c">12.9 GB</Value>
<Value name="c">0.5 tb</Value>
```

You can also use the old system using the suffixes like “kib”, “mib” and “gib”. This suffixes are using a factor of 1024, 1024² and 1024³. This is shown in the following example:

```
<Value name="a">128kib</Value>
<Value name="b">20 MiB</Value>
<Value name="c">177 GiB</Value>
```

If you specify a value using a byte amount suffix, you can also use fractions. The value 0.5 kb equals 500 byte. The suffix is case insensitive. Valid suffixes are: Byte, KB, MB, GB, TB, PB, EB,

ZB, YB and KiB, MiB, GiB, TiB, PiB, EiB, ZiB. YiB. The last two suffixes, *zetta-* and *yottabyte* can not be represented correctly in current systems. They exist to avoid problems with future versions of our software.

Boolean Type

The boolean type can only have one of the two states “true” or “false”. You can express this two states with the values `true` and `false`, or `yes` and `no`. These values are case insensitive.

In the next example, two boolean values are shown.

```
<Value name="a">false</Value>
<Value name="b">Yes</Value>
```

The Group Element

The Group element is a simple container to group multiple values in a configuration block together. A possible reason for a group is to make a group of values optional. But if you specify the group, you have to specify all values from this group.

A group accepts all type of value elements like the Module block does. All elements require the name attribute as well.

```
<Group name="a">
  <Value name="a">value1</Value>
  <Value name="b">value2</Value>
</Group>
```

The List Element

The List element is used for complex or simple lists. Complex means that each entry in the list contains a set of various values or even nested structures like a group does. A simple list is just a list with value elements.

Complex Lists

In a complex list, each element is put into a `ListEntry` block which has no name attribute. The definition for each list entry is the same for the whole list. There can also be some constraints on the list itself, notably a minimum or maximum number of elements.

The order of the elements in the list is preserved for the configuration and plays an important role in some cases for the application. A list of filters for example which is executed in the order of the definition or other such cases.

The following example shows a list with two entries. You can see how the definition is repeated for each entry, just the values change in the list entry.

```
<List name="x">
  <ListEntry>
    <Value name="a">12</Value>
    <Value name="b">test entry 1</Value>
  </ListEntry>
  <ListEntry>
    <Value name="a">20</Value>
    <Value name="b">test entry 2</Value>
  </ListEntry>
</List>
```

Simple Lists

If just a value is defined as element for a list, the list is composed from Value elements without name attribute.

The definition of the value entry is the same for each element. Also the order of the elements can be important in some cases for the application.

The following example is showing a simple list with five text values.

```
<List name="x">
  <Value>list entry A</Value>
  <Value>list entry B</Value>
  <Value>list entry C</Value>
  <Value>list entry D</Value>
  <Value>list entry E</Value>
</List>
```


Placeholders

You can use placeholder in each value tag. Placeholders are enclosed in two special Unicode characters (0x25CF) which represents the “back circle” character (●). The reason to use this very special character is to avoid any conflict with regular text. The “back circle” character is included in many fonts, so you should have no problem to find and use it.

There are two forms for the placeholder which are shown in the next example. The first form is just a placeholder with a given name and the second form takes some parameters in round brackets. The name of the placeholder is always *case sensitive*.

```
●name●  
●name(parameter)●
```

You can use placeholders in the content of every value in the configuration. Placeholder are working independently from the application.

The placeholder text is replaced at the application start, just after the configuration is read from the XML document. Therefore, dates and times are replaced with the date and time of the application start and are not changed while the application is running.

Application Directory Placeholder

The application directory placeholder is replaced by the directory from which the application was started.

```
●applicationDirectory●
```

The application directory never end in a directory separator. You have to add a directory separator if you like to add files or sub directories after this placeholder.

This Directory Placeholder

This placeholder is replaced with the directory from which the current configuration file is read.

```
●thisDirectory●
```

The replaced directory never ends in a directory separator.

Date/Time Placeholder

With this placeholder you can put the current date and/or time into the text. You can also optional adjust the current time by a given number of years, months, days or seconds. All parts of the placeholder are *case sensitive*.

The first format of this placeholder is shown in the next example. The **format string** is placed between two single quotes (Unicode 0x0027, "Apostrophe").

```
•datetime('format string')•
```

The second format is shown in the following example. After the **format string** an offset is added, which is a integer number followed by one letter which describes the unit of the offset.

```
•datetime('format string', offset unit)•
```

The possible expressions for the format string for the date/time is described in the section “[Date/Time Format](#)” on [page 22](#). The **offset** number can be any positive or negative integer number to adjust the date or time from the current date and time.

The **unit** of the adjustment is a single character:

C	Meaning
y	Years
m	Months
d	Days
s	Seconds

Some examples for the date/time placeholder are shown in the next example block.

```
•datetime('yyyyMMdd')•      ← will get replaced for example to 20110508
•datetime('yyyy')•          ← will get replaced for example to the year 2011
•datetime('yyyyMMdd', -12d)• ← current date minus 12 days.
•datetime('yyyy', 1y)•       ← the current year plus one.
```

Date/Time Format

The format string for dates and times can contain the expressions which are described in [Table 2 on page 23](#).

All other input characters will be treated as text. Any sequence of characters that are enclosed in single quotes will also be treated as text and not be used as an expression.

Expres- sion	Description
d	The day as number without a leading zero (1 to 31).
dd	The day as number with a leading zero (01 to 31).
ddd	The abbreviated localized day name (for example 'Mon' to 'Sun').
dddd	The long localized day name (for example 'Monday' to 'Sunday').
M	The month as number without a leading zero (1-12).
MM	The month as number with a leading zero (01-12).
MMM	The abbreviated localized month name (for example 'Jan' to 'Dec').
MMMM	The long localized month name (for example 'January' to 'December').
yy	The year as two digit number (00-99).
yyyy	The year as four digit number.
h	The hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display).
hh	The hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display).
H	The hour without a leading zero (0 to 23, even with AM/PM display).
HH	The hour with a leading zero (00 to 23, even with AM/PM display).
m	The minute without a leading zero (0 to 59).
mm	The minute with a leading zero (00 to 59).
s	The second without a leading zero (0 to 59).
ss	The second with a leading zero (00 to 59).
z	The milliseconds without leading zeroes (0 to 999).
zzz	The milliseconds with leading zeroes (000 to 999).
AP or A	Interpret as an AM/PM time. AP must be either "AM" or "PM".
ap or a	Interpret as an AM/PM time. ap must be either "am" or "pm".

Table 2 A table with all valid date and time format expressions.

Computer and Domain Name

There are two placeholders which you can use to use the computer or the domain name of the computer which is running the application.

- computerName•
- domainName•

Use Environment Variables

You can also use the value of a local defined environment variable as placeholder in your configuration.

- environmentVariable(temp)•
- environmentVariable(homedrive)•

The environment of the process which reads the configuration is used to get the specified value. If an environment variable does not exist, the placeholder is replaced with an empty string.

Log-On User Placeholders

The configuration system supports two special placeholders which are only available under Windows. With the following two placeholders you can use the user name or the domain of the logged in user which executes the application.

- userName•
- userDomain•

The user domain can be different from the domain name which is returned by the domainName placeholder.

Command Line Arguments

Applications which use the *Configuration B* system are usually supporting two special command line arguments which can change the way how the configuration files are read. They are explained in sections “[The Argument --configFile](#)” on page 26 and “[The Argument --configDirectory](#)” on page 26.

The system also supports command line arguments for the application itself. Command line arguments can be provided by any internal module and they are explained in the manual of the application.

There are short and long command line arguments. The format of this arguments is explained in the following sections.

Long Command Line Arguments

Long command line arguments start with two hyphens, followed by the *case sensitive* name of the argument.

The following example is showing long command line arguments which are used as flag, therefore they do not specify a value.

```
Application.exe --exampleArg  
Application.exe --arg1 --arg2 --arg3
```

If the command line argument requires a value it is specified after an equal sign as shown in the next example.

```
Application.exe --example=Simple  
Application.exe --maximum=12402  
Application.exe --another="Value with Spaces"  
Application.exe --arg1=100 --arg2=200
```

Short Command Line Arguments

Sometimes there is a short version for long command line arguments. A short command line argument starts with one single hyphen followed by one letter.

The following example is showing some short command line arguments used as flags. Therefore they do not specify any value.

```
Application.exe -a  
Application.exe -x -v -z -b
```

If the command line argument requires a value it is specified after an equal sign as shown in the next example.

```
Application.exe -a=Simple  
Application.exe -a="Value with Spaces"  
Application.exe -x=1023 -v=12 -z=Test
```

You can mix long and short forms of command lines. The short forms are usually just provided for convenience.

The Argument --configFile

You can use the `--configFile` command line argument multiple times to set a different primary configuration file or to add additional configuration files.

Each specified file is read and parsed by the configuration system. If one of the specified files not found, an error is displayed and the start of the application is stopped.

The following example specifies an alternate configuration file for the fictive application.

```
Application.exe --configFile=X:/Text/myConfig.xml
```

The Argument --configDirectory

This argument adds an additional search directory to the list of the default directories. It is searched for configuration files with the suffix `.conf_b.xml` or for files called `configuration.xml`.

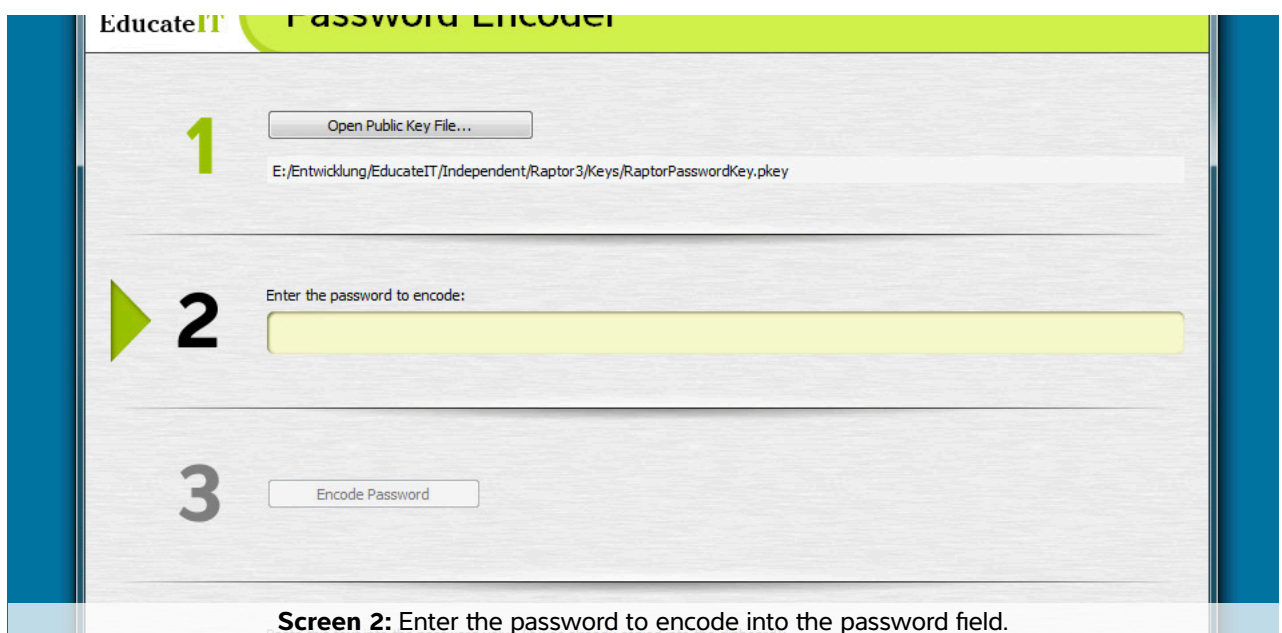
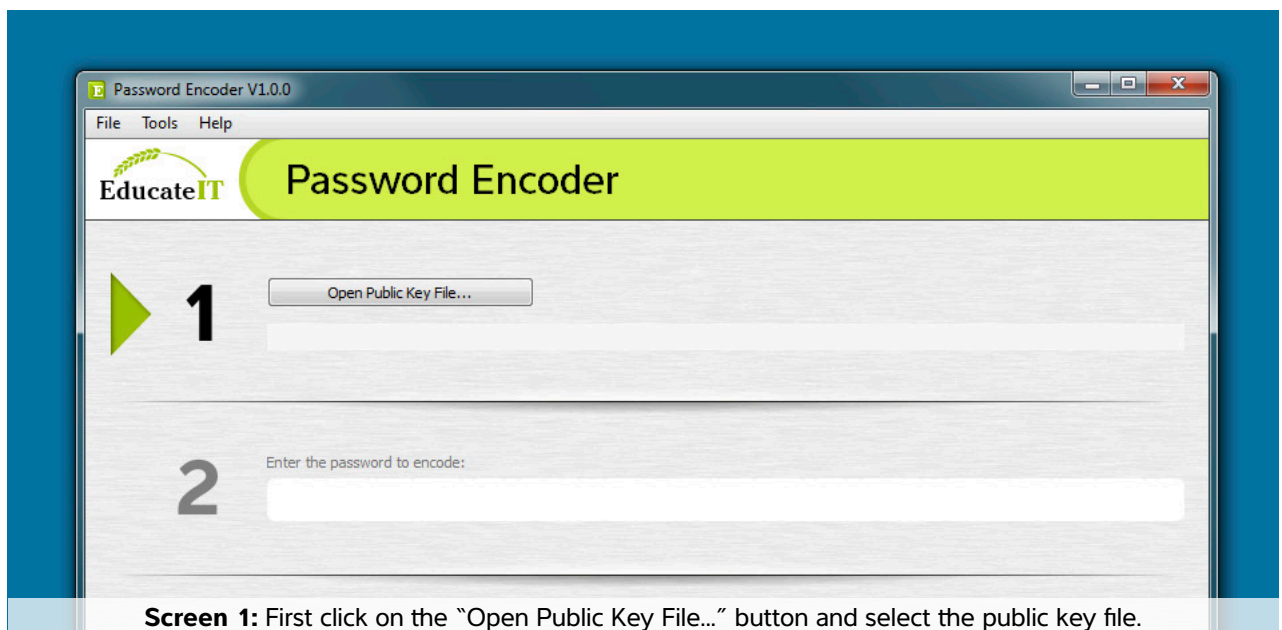
The following example specifies an additional search directory which is searched for configuration files.

```
Application.exe --configDirectory=X:/Config
```

Password Encoding

The configuration system is supporting encoded passwords. An encoded password is encrypted for the application which makes it very hard to decrypt and read them.

To encode passwords you need the “EducateIT Password Encoder” application and the public key provided by the application for which you encode the password.



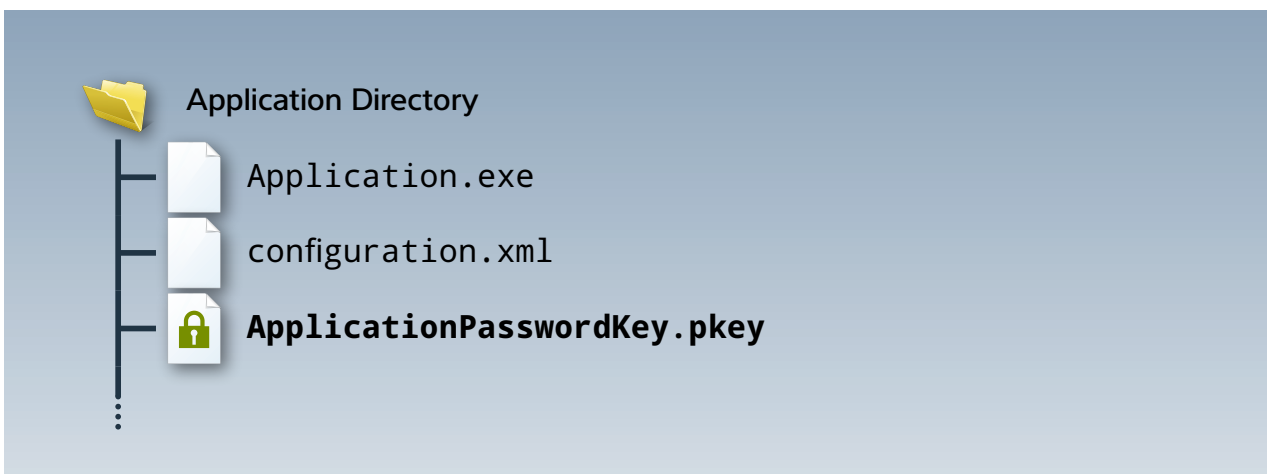
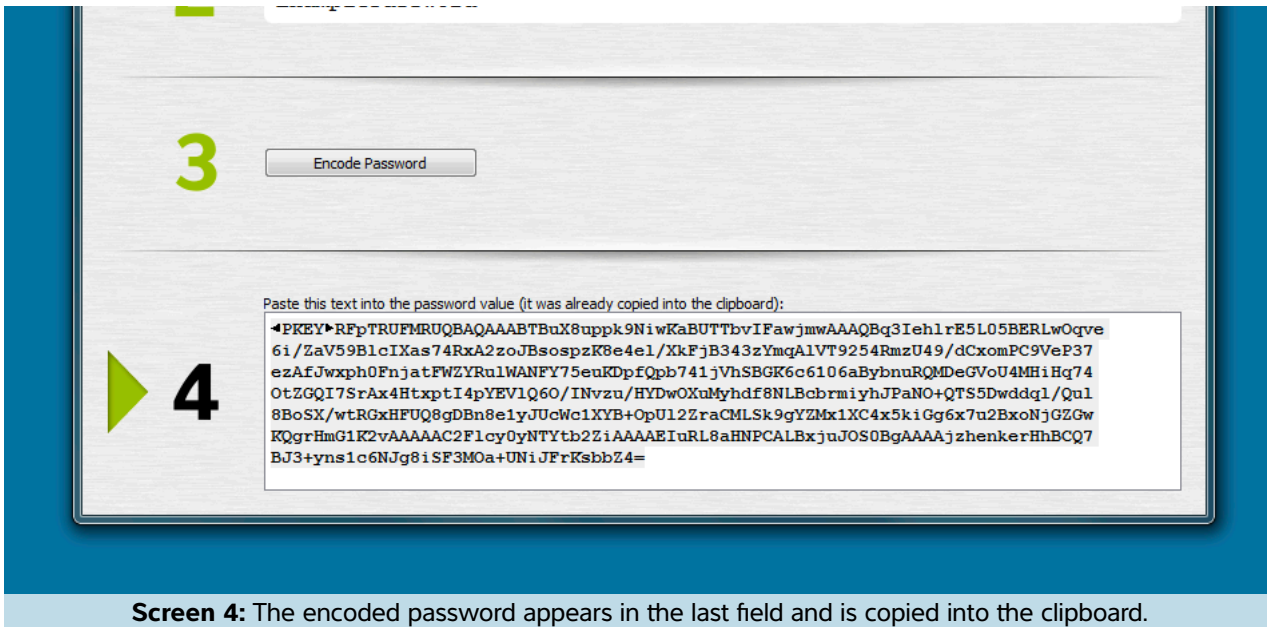
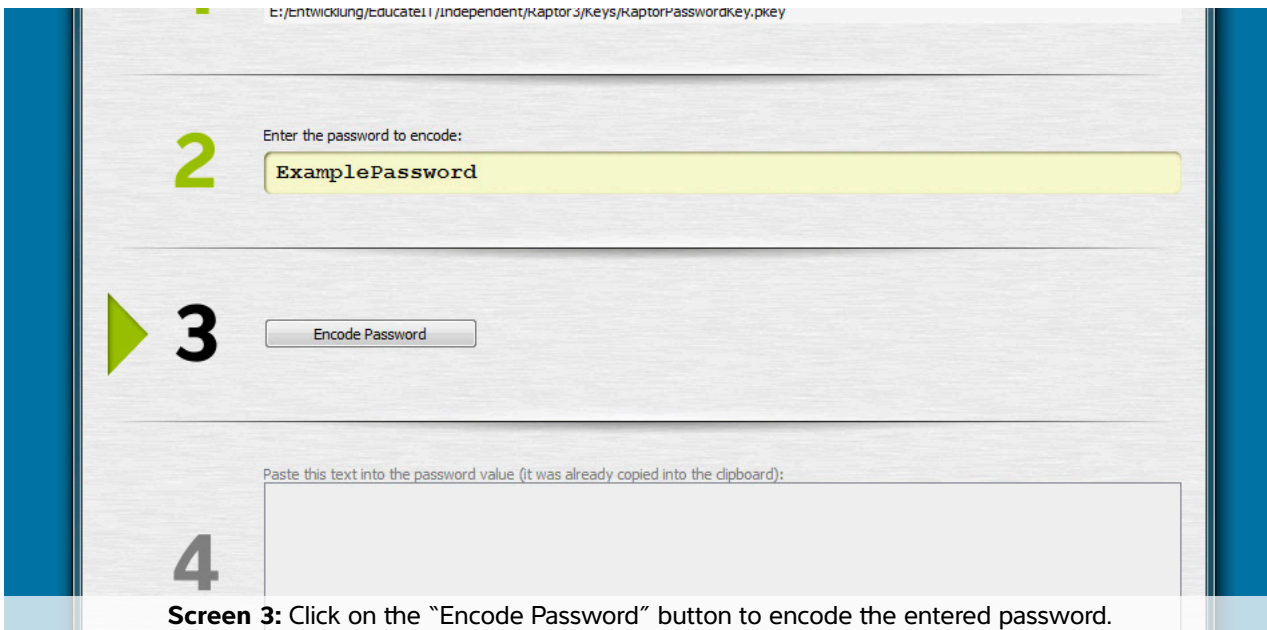


Illustration 5 The password public key is located in the same directory as the application binary.

Step 1: Locate the Public Key

The public key for password encoding is located in the same directory as the application binary and has the suffix .pkey. See [Illustration 5 on page 28](#). The name of the key usually starts with the name of the application.

If this key is missing, the application probably does not support password encoding.

Step 2: Encode the Password

To encode the password start the EducateIT Password Encoder application. Click on the “Open Public Key...” button. Now select the public key file of the application. If the key was loaded with success, the green arrow moves to the next step. See [Screen 1 on page 27](#) to [Screen 4 on page 28](#) how it works.

Enter the password you like to encode into the password field. If you entered a password click on the button “Encode Password” to start the encoding.

The encoded password appears in the last field at step four of the application. It is copied into the clipboard as well.

Step 3: Paste It Into The Configuration

Now, you paste the encoded password into the configuration at the place where the password is configured. See [Listing 4 on page 29](#) for an example

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration ...>

  <Module name="ActiveDirectory">
    <List name="ServerList">
      <ListEntry>
        <Value name="Label">Server 1</Value>
        <Value name="Host">example1</Value>
        <Value name="UserName">administrator</Value>
        <Value name="Password">
          ◀PKEY▶RFpTRUFMRUQBAQAAABTBuX8uppk9NiwKaBUTtbvIFawjmwAAAQDJPHiIHsaE+81DyJ1NgpD3
          o/1Fz87NtNk7QmEAYOo3BcBQ8yf+0LnHqFWv8AilCjxULZSu/m1bsdnPrQbTSZ99CHXmfszWfcokR
          BwCj16iCB6pZXUKMvpMTAvfuUxVp1fPwNp1teATVmwBDz1g7hoAjkWC4Bd8crVXRi8dPqP7k5yS+7
          yCdXo4Dc3H0ZqrMS1f19J1nDshA+U5NQQDhaE5B7vMmkCpzL7vYa5HmRMxfYuQTF1mYeL5EOr1X9P
          WeFxp9eXI1L/FZUH+7J3HZVv0BWWBz65e5y/dQPIXv0UF5zeaS9bLAWDzHWyG1H17aGft+HUSNDO
          QC8sqlGrAQ2IAAAAC2Flcy0yNTYtb2ZiAAAAEG6pMww0CduAMj8SqACEDXQAAAAciMPQybnOXK0Ba
          QGj6PTMb21yy1h2vU1FrSUY2Q==
        </Value>
      </ListEntry>
    </List>
  </Module>
  ...
</Configuration>
```

Listing 4 An encoded password pasted into the configuration.

The encoded password contains Unicode characters. Make sure your editor is set to the correct encoding for the configuration file. Usually configuration files are encoded with *UTF-8*.

How Secure are Encoded Passwords?

We are using strong encryption to protect the encoded passwords. If one just get access to the configuration file alone, it is impossible to uncover the password.

The system uses asynchronous encryption. First the password is encrypted with the public key, which is provided as a file outside of the application. The private key is embedded in the application itself.

The embedded private key is the biggest weakness of the system. Naturally the key is embedded in a very secure way and really hard to extract, nevertheless with access to the application binary, vast amount of time and very high software development skills, it is possible to extract the private key and decrypt the password.

Therefore the encoded passwords are a good way to prevent employees easily access the passwords by reading the configuration files, but it is not a protection against highly specialized hackers.

Security Recommendations

- Limit access to the binary and configuration file if possible. For example in a multi-tier system make sure only administrators and the account under which the server is running can access the server binary and configuration file.
- Never store passwords of accounts which are actively used by human operators.
- If you need additional privileges on remote systems, create a separate account with a own password for the system. In case the password is uncovered you can easily track and block the access with this account. Also limit the privileges of this account to the minimum possible. If possible even limit access of this account to just the system from which the access should come.
- Keep an eye on the protection of test systems as well. Just because they are not used productively they can be a the entry vector for an attack of your system.

Configuration Visualization

In our newer manuals we use a unified visual format to describe the constraints and values of the configuration values. In this section I am describing the different visual elements of this visualization which will allow you to easily read and understand this illustrations.

All the elements shown in [Illustration 6](#) on [page 31](#) are explained in [Table 3](#) on [page 32](#).

Basic Division

There are two main columns. The column on the left side names the type of element on the right side. The right side contains the elements itself.

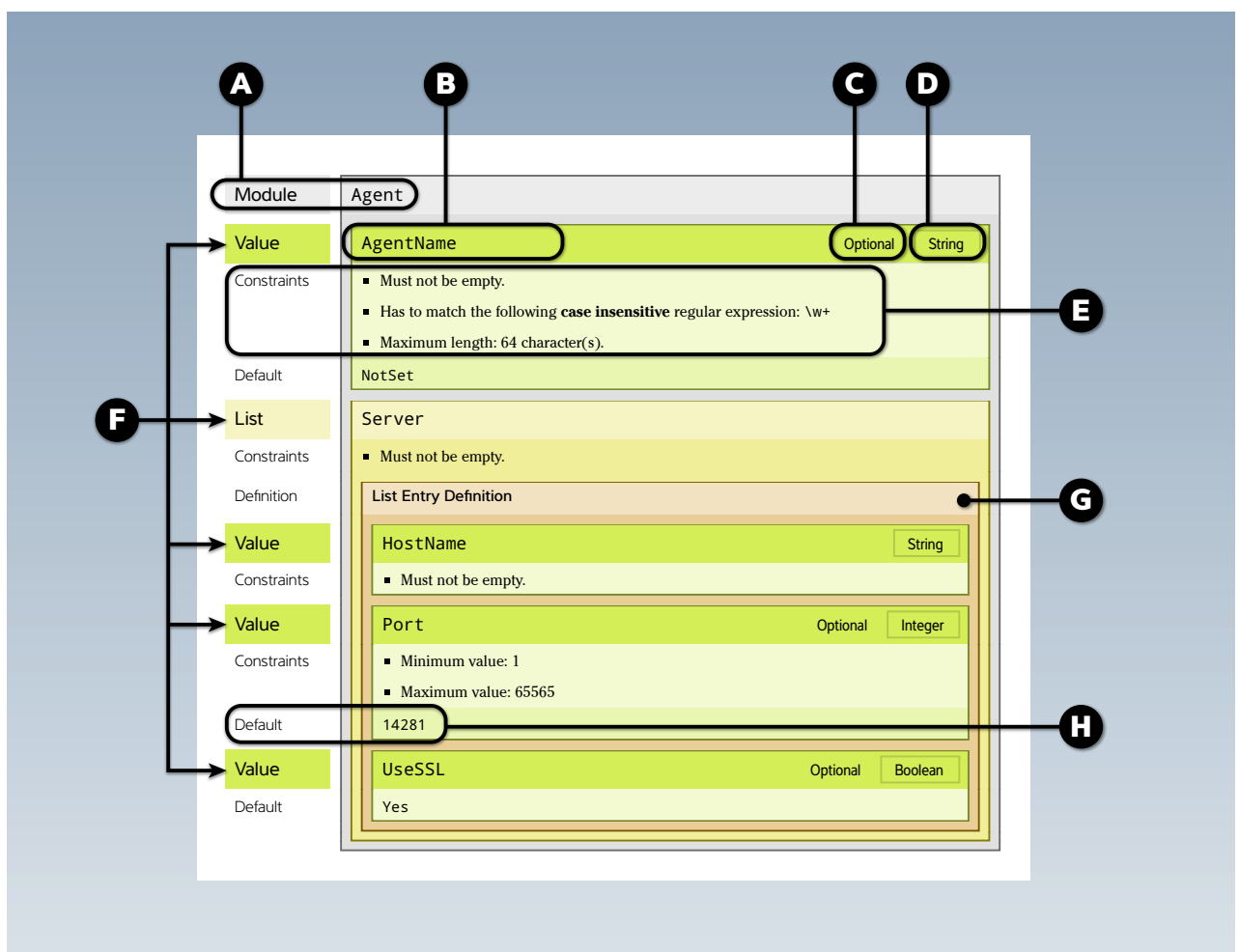


Illustration 6 The different elements of the configuration visualization.

Letter	Description
A	The name of the configuration module is shown in the first line of the visualization. In this example, the name of the module is “Agent”.
B	Each element inside of the configuration module block has a name. This name is written in the first row of each box in the visualization. There are special cases for the definition of a list entry or the value of a value list.
C	This flag is shown if a value is optional. An optional value can be omitted. If you omit an optional value, the default value is used if one is defined.
D	Value elements can have different types as explained in “ The Types of the Value Element ” on page 16. The type of the value created some implicit constraints. Obviously an Integer value for example can not contain random text.
E	There are additional constraints in the block labelled with “Constraints”. The value has to match or fulfil each constraint in the list. The application can have constraints which are not shown in the visualization, like constraints which require a value to be unique in a list.
F	The basic type of the configuration value is shown in the left column. It is valid for the whole box on the right side. This type is equal to the tag name in the configuration file. If you see a List type in this column it is required you insert a <List ...> tag in the configuration file. See “ Configuration File Structure ” on page 11 for details about the different configuration value types.
G	For lists and groups there are special definition boxes. They enclose the definition for one single entry for a list or the definition of the contents of a group.
H	An optional value can have a default value. If the optional value is omitted in the configuration, the default value is used. This behaviour can be changed by the application, make sure you read the details in the application manual.

Table 3 The descriptions of all elements of the visualization.

```

<Module name="Agent">

  <List name="Server">
    <ListEntry>
      <Value name="HostName">192.168.10.4</Value>
      <Value name="Port">14281</Value>
    </ListEntry>
    <ListEntry>
      <Value name="HostName">192.168.20.5</Value>
    </ListEntry>
  </List>

  <Value name="AgentName">NotSet</Value>

</Module>

```

Listing 5 An actual configuration made from the example visualization.

Boxes on the right side are showing you the nesting of the elements. A box is an element with a border around. The outermost gray box represents the module block itself. Each box represents a nested tag in the configuration file. If a box is nested in another box, the tag in the configuration is nested in this tag as well.

See [Listing 5 on page 32](#) for an example how the final configuration module could look like.

[Illustration 7 on page 33](#) shows you the mapping between the elements in the visualization and the elements in the configuration file.

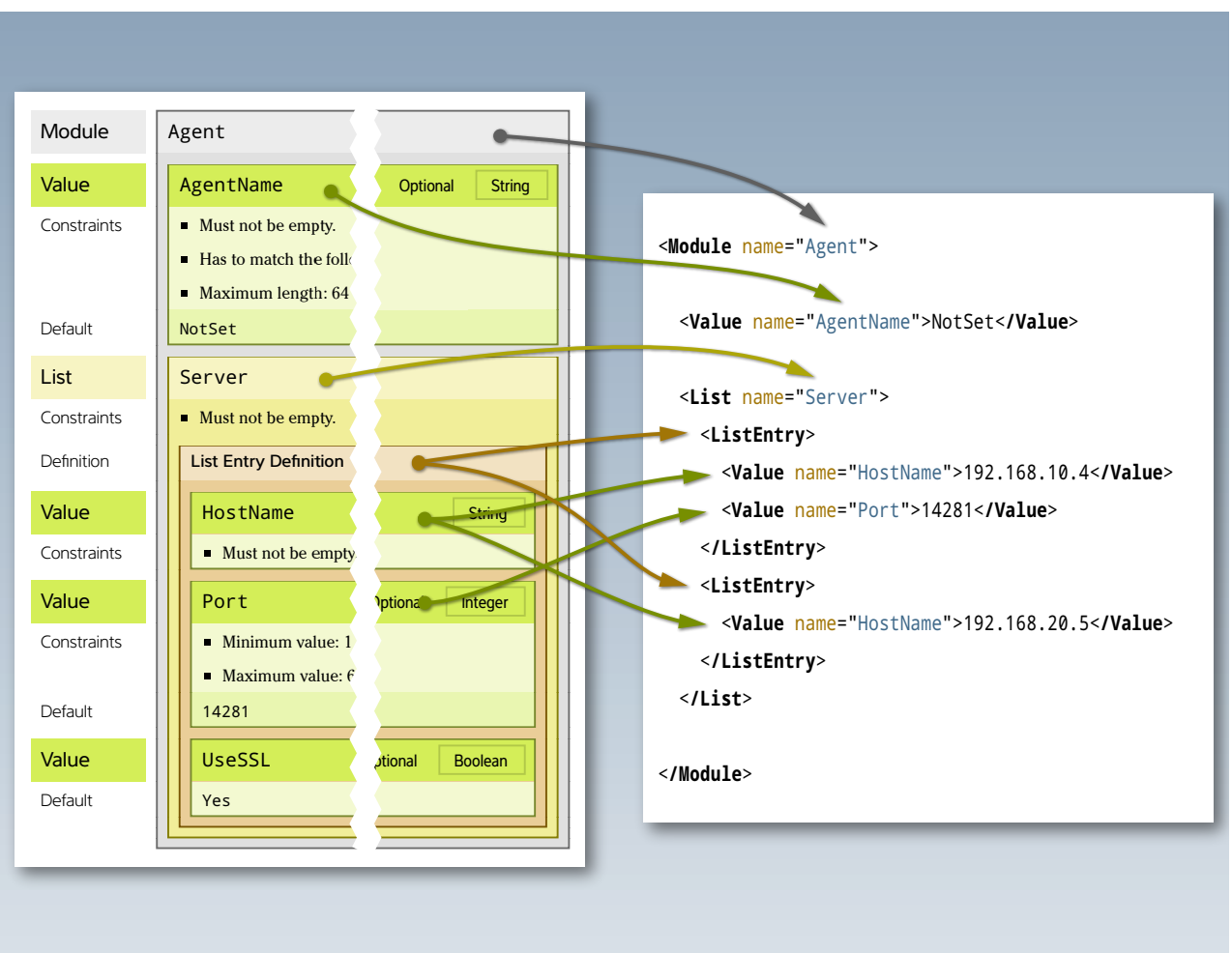


Illustration 7 The mapping between the visualization and the configuration file.

Derived Host Configuration

Some multi-tier systems provide the feature of a derived host configuration. This allows you to provide different versions of module configuration blocks depending on the host which requests the configuration from the server. Multiple hosts can be joined to a group which has a unique name. This name is used in the configuration file to define the derived configuration module blocks.

The HostGroups Block.

At the begin of the primary configuration file a HostGroups block is required. This block can contain any number of host group definitions. See [Listing 6 on page 35](#) for an example.

The HostGroup Definition

Each host group definition requires a name attribute. The defined name has to be unique for the whole configuration file. It can consist of letters, numbers, hyphens and the underscore character. The maximum length is 64 characters.

There can be any number of HostAddress and Network blocks inside of a host group definition. They are explained in the following sections.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration version="1"...>

  <HostGroups>
    <HostGroup name="example1">
      <HostAddress>127.0.0.1</HostAddress>
      <Network>192.168.12.0/24</Network>
    </HostGroup>

    <HostGroup name="example2">
      <HostAddress>fe80::4d23:5fa6:f8eb:eaca</HostAddress>
      <Network>192.168.19.0/16</Network>
    </HostGroup>
  </HostGroups>

  ...

</Configuration>
```

Listing 6 A configuration file with derived host configurations.

Single Host Definition With the HostAddress Tag

You can define a single *IPv4* or *IPv6* address with the Host-Address tag. See the following examples:

```
<HostAddress>127.0.0.1</HostAddress>  
<HostAddress>10.127.14.9</HostAddress>  
<HostAddress>fe80::4e23:5fa6:f8eb:eaca</HostAddress>  
<HostAddress>2001:0DB8:0:CD30::1</HostAddress>
```

Network Definitions With the Network Tag

With the Network tag you can define whole networks. You can use the *IPv4* or the *IPv6 CIDR* format. See the following examples:

```
<Network>192.168.12.0/24</Network>  
<Network>10.9.190.1/12</Network>  
<Network>2001:0DB8:0:CD30::1/60</Network>
```

How Hosts are Assigned to Groups

To assign a given host to a group, the list is checked in the order in which it is defined. Each host address and network is checked if it matches the given host address.

If a match is found, the name of the group of the match is assigned to the host. In case there was no match in the whole list, no group name is assigned to the host.

Derived Configuration Blocks

To create a special configuration for a defined host group you add a derived module configuration block to the configuration file. You always need a regular module configuration block from which you create the derived configuration.

See [Listing 7 on page 37](#) for an example of a derived module configuration block. It uses the same name as the regular module configuration block but adds the `hostGroup` attribute with the name of one of the defined host groups.

You can “overwrite” any of the top level values from the regular block by defining a new value with the same name. All values you omit are taken from the regular block.

In the example listing a host which matches host group “example1” will get the following configuration values:

```
a = 100  
b = 200  
c = 770
```

```

<?xml version="1.0" encoding="UTF-8" ?>
<Configuration version="1"...>

    ...

    <Module name="demo">
        <Value name="a">100</Value>
        <Value name="b">200</Value>
        <Value name="c">500</Value>
    </Module>

    <Module name="demo" hostGroup="example1">
        <Value name="c">770</Value>
    </Module>

</Configuration>

```

Listing 7 A derived module configuration block.

Any other host will get this configuration values:

```

a = 100
b = 200
c = 500

```

You can create one derived configuration module block for each defined host group.

Example

See [Listing 8 on page 38](#) for a fictive complete example of a configuration file. It defined four different host groups and creates derived configuration module blocks for the “Demo” module.

```

<?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">

  <HostGroups>
    <HostGroup name="example1">
      <HostAddress>127.0.0.1</HostAddress>
      <Network>192.168.12.0/24</Network>
    </HostGroup>

    <HostGroup name="example2">
      <HostAddress>fe80::4d23:5fa6:f8eb:eaca</HostAddress>
      <Network>192.168.19.0/16</Network>
    </HostGroup>

    <HostGroup name="example3">
      <HostAddress>192.168.48.11</HostAddress>
    </HostGroup>

    <HostGroup name="example4">
      <Network>192.168.48.0/24</Network>
    </HostGroup>
  </HostGroups>

  <Module name="Network">
    <Value name="KeepAliveInterval">10</Value>
    <Value name="KeepAliveTimeout">15</Value>
    <Value name="MinimumAgentVersion">3.0.0</Value>
    <Value name="BlockAgentWithSameId">Yes</Value>
    <Value name="LogAgentResponseTimeInterval">60</Value>
  </Module>

  <Module name="Demo">
    <Value name="a">100</Value>
    <Value name="b">200</Value>
    <Value name="c">500</Value>
    <List name="things">
      <Value>Entry1</Value>
      <Value>Entry2</Value>
    </List>
  </Module>

  <Module name="Demo" hostGroup="example1">
    <Value name="c">770</Value>
  </Module>

  <Module name="Demo" hostGroup="example2">
    <Value name="a">90</Value>
    <Value name="b">300</Value>
    <Value name="c">800</Value>
  </Module>

  <Module name="Demo" hostGroup="example4">
    <Value name="c">900</Value>
    <List name="things">
      <Value>Entry1</Value>
    </List>
  </Module>

</Configuration>

```

Listing 8 A complete example of a configuration file with derived configuration module blocks.

Migration From Base System

If you are using a product which changed the configuration system from “Base System” to the new “Configuration B” system, but the configured content did not change, then a conversion from the old to the new format is quite simple.

All what you have to do is to add the correct XML schema to the document tag and change the names of the XML elements slightly. The following table is showing the changed element names.

Old Element Name	New Element Name
configuration	Configuration
module	Module
value	Value
list	List
listItem	ListEntry
group	Group

An old “Base System” document looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <module name="Application">
    <value name="example">Example Value</value>
  </module>
</configuration>
```

It has to be converted into this:

```
<?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="example">Example Value</Value>
  </Module>
</Configuration>
```

As you can see, the value names and the value content is untouched and do not need any conversion.

Automate Conversion With Perl

You can use the *Perl* script shown in [Listing 9 on page 40](#) to convert a configuration file from the “Base System” format, into the new format of the “Configuration B” system.

You pass the old configuration file as argument to the script and it will return the converted script to the console.

```
Converter.pl OldConfig.ini > configuration.xml
```

```
#!/usr/bin/perl -w
#
# Configuration Converter - V1.0.0
# =====
# This script is reading a configuration in the old format and converting it
# into a configuration in the format required by the “Configuration B”
# System.
#
# Usage:
#   ConfigurationConverter.pl [Old Configuration File] > configuration.xml
#
use strict;

#
# Main conversion loop
#
while( <> )
{
    my $line = $_;

    $line =~ s/\<configuration>/\<Configuration\n\tversion=\"1\"\\n\\
\txmlns=\"http://educateit.ch/software/BlueStone/Configuration/1\"\\n\\
\txmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\">/g;
    $line =~ s/\<(\\?)listItem/\<\1ListEntry/g;
    $line =~ s/\<(\\?)(module|list|group|value|configuration)/\<\1u\2/g;

    print( $line );
}

# =====
# End of File
#
```

Listing 9 Perl script to convert the “Base System” configuration to the new format.

Configuration File Schema

For your reference, the schema for the configuration files is printed on this and the next page. You should get it together with the software product. It was probably already installed into the application directory, named “ConfigurationB.xsd”. Otherwise get in touch with us.

```
<?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/BlueStone/Configuration/1"
  xmlns="http://educateit.ch/software/BlueStone/Configuration/1"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="HostGroupName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[_a-zA-Z0-9]+" />
      <xs:minLength value="1" />
      <xs:maxLength value="64" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ValueName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[_a-zA-Z0-9]+" />
      <xs:minLength value="1" />
      <xs:maxLength value="64" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="ModuleName">
    <xs:restriction base="xs:string">
      <xs:pattern value="[_a-zA-Z0-9]+" />
      <xs:minLength value="1" />
      <xs:maxLength value="64" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="PlaceholderFlag">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Replace">
        <xs:annotation>
          <xs:documentation>Replace all placeholders in the text of this value.</
xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="Ignore">
        <xs:annotation>
          <xs:documentation>Ignore all placeholders in the text of this value.</
xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
```

```

        </xs:enumeration>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="Value">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="placeholder" type="PlaceholderFlag" use="optional"
default="Replace" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="NamedValue">
    <xs:simpleContent>
        <xs:extension base="Value">
            <xs:attribute name="name" type="ValueName" use="required" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="Group">
    <xs:sequence>
        <xs:group ref="Elements" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="NamedGroup">
    <xs:complexContent>
        <xs:extension base="Group">
            <xs:attribute name="name" type="ValueName" use="required" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="NamedList">
    <xs:choice>
        <xs:element name="ListEntry" type="Group" minOccurs="0" maxOccurs="unbounded" />
        <xs:element name="Value" type="Value" minOccurs="0" maxOccurs="unbounded" />
    </xs:choice>
    <xs:attribute name="name" type="ValueName" use="required" />
</xs:complexType>

<xs:group name="Elements">
    <xs:choice>
        <xs:element name="Value" type="NamedValue" />
        <xs:element name="Group" type="NamedGroup" />
        <xs:element name="List" type="NamedList" />
    </xs:choice>
</xs:group>

<xs:complexType name="Module">
    <xs:sequence>
        <xs:group ref="Elements" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="ModuleName" use="required" />
    <xs:attribute name="hostGroup" type="HostGroupName" use="optional" />
</xs:complexType>

<xs:complexType name="HostGroup">
    <xs:sequence>
        <xs:choice minOccurs="1" maxOccurs="unbounded">
            <xs:element name="HostAddress" type="xs:string" />

```

```

        <xs:element name="Network" type="xs:string" />
    </xs:choice>
</xs:sequence>
<xs:attribute name="name" type="HostGroupName" use="required" />
</xs:complexType>

<xs:complexType name="HostGroups">
    <xs:sequence>
        <xs:element name="HostGroup" type="HostGroup" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="Configuration">
    <xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Module" type="Module" />
            <xs:element name="IncludeFile" type="xs:string" />
            <xs:element name="IncludeDirectory" type="xs:string" />
            <xs:element name="HostGroups" type="HostGroups" />
        </xs:choice>
    </xs:sequence>
    <xs:attribute name="version" type="xs:integer" use="required" />
</xs:complexType>

<xs:element name="Configuration" type="Configuration" />
</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/>