



Configuration B

Administrator Manual

Software Version 1.1.0

EducateIT GmbH

Phone: +41 32 355 39 00

E-Mail: info@educateit.ch

Address: Steinenbergstrasse 7, 4532 Feldbrunnen, Switzerland

Contents

Preface	5
Features	5
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	11
The IncludeFile tag	12
The Module tag	12
The Value Element	14
The Group Element	15
The List Element	15
The ValueList Element	16
Placeholders	17
Application Directory Placeholder	17
This Directory Placeholder	17
Date/Time Placeholder	18
Date/Time Format	18
Computer and Domain Name	19
Migration From Base System	20
Automate Conversion With Perl	21
Configuration File Schema	22

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 products which are using this configuration system. You probably got this document along with the manual of the software, for the reason that the 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 if you are reading the correct document. See section “[How to Find the Version Number](#)” 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.

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 our company.

Configuration Modules

For huge applications, each part is configured in a own section of the configuration. That way you can reuse existing configurations

for certain modules, if you worked with an application which used the same component before.

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.

Placeholders for Paths and Dates

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 be used to automatically build paths and file-names for configuration.

How to Find the Version Number

If you got this manual together 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.



Illustration 1 The about window.

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 [Illustration 1](#) for an example how the about screen looks like.

The detail pages summarize various informations about the internal structure of the application. Search for an 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** 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.

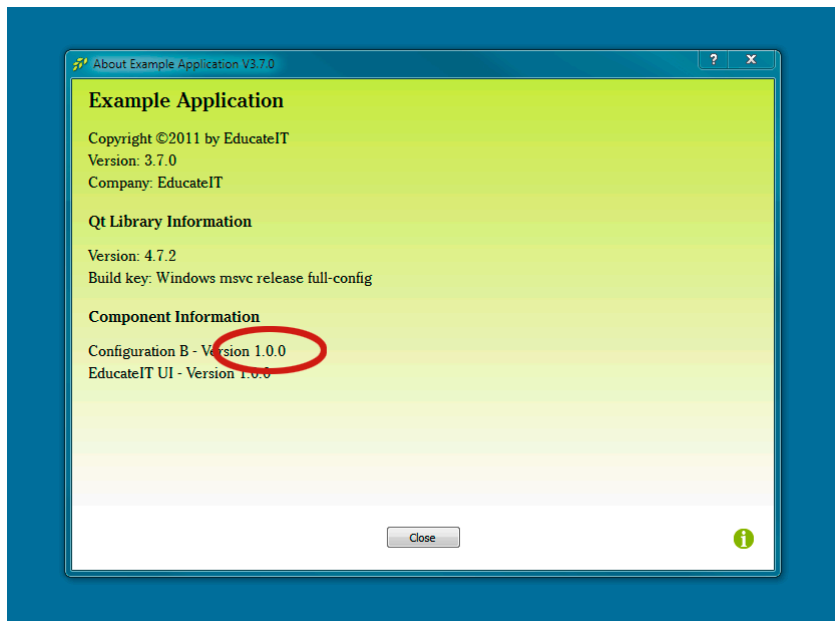


Illustration 2 The detail page of the about window.

Files and Locations

There is a default behaviour where the system is searching for configuration files. But please note, that 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.

All found configuration files are read and parsed. After this, the values are verified and assigned.

OS	Version	Path
Mac	OS X	/Library/Application Support/EducateIT/ <i>Product Name</i>
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/ <i>Product Name</i> /opt/etc/EducateIT/ <i>Product Name</i> /usr/etc/EducateIT/ <i>Product Name</i> /etc/EducateIT/ <i>Product Name</i>
Windows	97 XP 2000	C:\Documents and Settings\All Users\Application Data\EducateIT\ <i>Product Name</i> %ALLUSERSPROFILE%\Application Data\EducateIT\ <i>Product Name</i>
	7	C:\ProgramData\EducateIT\ <i>Product Name</i> %SystemDrive%\ProgramData\EducateIT\ <i>Product Name</i>

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

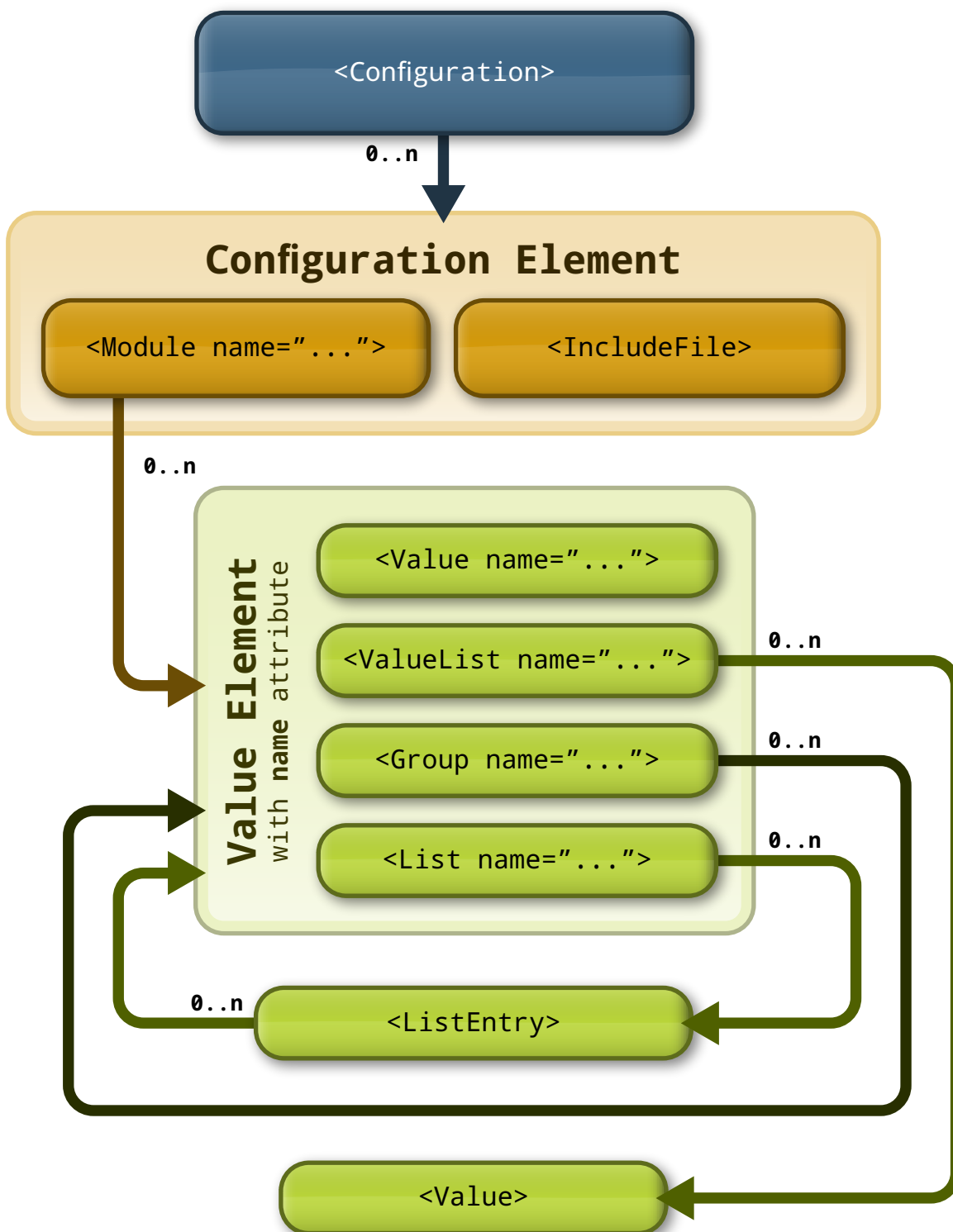


Illustration 3 Structure of the configuration file.

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 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 box,

```
<?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.

labelled `<Configuration>`, represents the same tag as shown in **Listing 1**. 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.

The Configuration Elements

Inside of the `Configuration` tag, the `Module` and the `IncludeFile` 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 is showing 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 were 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 location. It requires as content an absolute path to the file which you wish to include. This file has to be in the same format and 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.

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

The Module tag

The configuration itself is built 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 this elements can get nested.

```
<?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.

Each module block refers to a certain configuration module inside of the 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.

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 shows a fictive example of a module block. It contains all four possible value entries: Value, Group, List and ValueList. They are explained in detail in the next sections.

```
<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>

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

</Module>
```

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

The Value Element

The Value element actually contains some configuration value. It contains text, which can have some constraints on it. For example a configuration value can require an integer, 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.

There are a number of basic configuration types, which are stated in the application manual. They have some constraints on it, which are described in the next sections. But there can be more constraints, like a minimum or maximum value or length of a text. Or a text which has to be an element in a fixed enumeration. All this should be part of the description in the application manual.

String Type

The string type can be any kind of text. In 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 length.

String List Type

A string list is in fast 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 like 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>
```

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. This 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 lists. Complex means that each entry in the list contains a set of various values or even nested structures like a group does.

In the 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 in some cases an important role for the application. For example a list of filters which is executed in the order of definition and similar 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.

```
<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>
```

The ValueList Element

The ValueList element is similar to the list element, but is supporting only one value per entry. As for the List element, the order of the elements is preserved and can have a meaning to the configured application. Note that there is no name attribute for the Value tag.

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

```
<ValueList 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>
</ValueList>
```

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 independent 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”. 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 19](#).

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.

Expression	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•

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 4](#) 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 4 *Perl* script to convert the “Base System” configuration to the new format.

Configuration File Schema

For your reference, the schema for the configuration file is printed on this and the next page. You should get it together with the software product. It was probably installed into the application directory, called "ConfigurationB.xsd". Otherwise request it from our company.

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

  <xs:complexType name="ValueListValue">
    <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="Group">
```

```

<xs:sequence>
  <xs:group ref="Elements" minOccurs="0" maxOccurs="unbounded" />
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="List">
  <xs:sequence>
    <xs:element name="ListEntry" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:group ref="Elements" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="ValueList">
  <xs:sequence>
    <xs:element name="Value" type="ValueListValue" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
</xs:complexType>

<xs:group name="Elements">
  <xs:choice>
    <xs:element name="Value" type="Value" />
    <xs:element name="Group" type="Group" />
    <xs:element name="List" type="List" />
    <xs:element name="ValueList" type="ValueList" />
  </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="xs:string" use="required" />
</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: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/>

Author: Tobias Erbsland
Layout & Design: Tobias Erbsland
Last Modification: 2011-10-31 23:13
File Name: Configuration B - Administrator Manual
Copyright: ©2011 by EducateIT GmbH