

Ultimate++ Package Templates

0. Contents

1. Prologue
2. Description
3. Ready-to-use Templates
 - 3.1 Overview
 - 3.2 Basic CtrlLib application
 - 3.3 Console application (no U++)
 - 3.4 Core console project
 - 3.5 CtrlLib application with main window
 - 3.6 SDL project (no U++)
 - 3.7 SDL with Core package
4. Template File Layout
 - 4.1 Extension
 - 4.2 Sections
 - 4.3 Section Order
 - 4.4 Size
5. Syntax
 - 5.1 Comments
 - 5.2 White Space
 - 5.3 Basic Rule
 - 5.4 Delimiters
 - 5.4.1 "@@"
 - 5.4.2 "??"
 - 5.5 Expressions
 - 5.5.1 Basic Rule
 - 5.5.2 "@@filename_expression"
 - 5.5.3 "?expression"
 - 5.5.4 "<: ... :>"
 - 5.5.5 "<:? expression :> ... <.:>"
 - 5.5.6 "<:? expression :> ... <./:> ... <.:>"
 - 5.6 "PACKAGE"
 - 5.6.1 Description
 - 5.6.2 Usage
 - 5.7 "PACKAGE_UPPERCASE"
 - 5.7.1 Description
 - 5.7.2 Usage
 - 5.8 Header Line
 - 5.8.1 Description
 - 5.8.2 "template"
 - 5.8.3 "Name"
 - 5.8.4 "main", "sub"
 - 5.8.5 Usage
 - 5.9 "filename"
 - 5.9.1 Description
 - 5.9.2 Definition
 - 5.9.3 Usage
 - 5.10 "id"
 - 5.10.1 Description
 - 5.10.2 Definition
 - 5.10.3 Usage
 - 5.11 "option"
 - 5.11.1 Description
 - 5.11.2 Definition
 - 5.11.3 Usage
 - 5.12 "select"

	5.12.1 Description
	5.12.2 Definition
	5.12.3 Usage
	5.13 Undefined variables
6.	Template Dialog Layout
7.	Deployment
8.	Epilogue
Appendix: UPP File Syntax	

1. Prologue

Often projects have the same base, the same structure, and / or even identical parts of code. One way to avoid unnecessary typing or similar work when creating a new project like that, is to use a parameterized template.

An Ultimate++ package template (UPT) is such a parameterized template. It is used to create one or more package files which can be edited further. There can be more than one UPT.

And you can create a new UPT yourself! To enable you to do this, is the main purpose of this document.

2. Description

A UPT is a special file with a .upt-extension containing text and rules to create additional text.

When starting TheIDE or calling "File / Set main package..", the appearing window shows a "New Package" option. If this option is selected, TheIDE looks for UPTs and presents them in a list. Selecting one of them automatically shows a preview of the file(s) to create, possibly modified by options appearing at the same time. When everything is as wanted, "Create" writes the previewed files to disk and the new project is ready to be worked on.

3. Ready-to-use Templates

3.1 Overview

Ultimate++ is shipped with the following six UPTs:

- Basic CtrlLib application
- Console application (no U++)
- Core console project
- CtrlLib application with main window
- SDL project (no U++)
- SDL with Core package

"<empty>" means: do not use any package template.

3.2 Basic CtrlLib application

This UPT builds only the foundations of a GUI application: the needed include directive, the GUI equivalent for "main()", and the required build flag. You have the options to generate a header file (with include guards) and to add the "multithreaded" build flag.

3.3 Console application (no U++)

This UPT creates the foundations of a plain C++ console application. Even the non-GUI facilities of Ultimate++ cannot be used. There aren't any options.

This template comes in handy when you just want to exploit TheIDE.

3.4 Core console project

This UPT builds the foundations of a console application which uses only the non-GUI facilities of Ultimate++. You have the options to create a header file (with include guards) and to add the "multithreaded" build flag. Additionally you can direct the template to include code which reads the command line arguments into a const Vector<String>. (Please note that this is not the same as char* argv[], the "pointees" of which are modifiable and include an additional string - argv[0], the path and name of the running program.)

3.5 CtrlLib application with main window

This seems to be the most useful UPT which lays the foundations of a "big" GUI application. You get files containing the framework of your own application class, the name of which you can choose. You have a layout menu offering: no layouts, a general layout file, a main window-specific layout file, and a main-window-specific layout file generating an "OK"- and a "Cancel"-Button. Finally there is an option to generate an image list file and an option to add the "multithreaded" build flag.

3.6 SDL project (no U++)

This UPT generates the foundations of a Simple DirectMedia Layer (SDL) console application. The Ultimate++ facilities cannot be used. Apart from an option to create a header file and an option to write an event loop, there are three more multimedia-specific options .

This template comes in handy when you just want to exploit TheIDE.

3.7 SDL with Core package

This UPT builds the foundations of a Simple DirectMedia Layer (SDL) console application. Additionally, the non-GUI Ultimate++ facilities are enabled. Apart from an option to create a header file and an option to write an event loop, there are three more multimedia-specific options .

4. Template File Layout

4.1 Extension

The extension of a UPT file must be ".upt". If a file has a different extension or none at all it is not even considered to contain a UPT.

4.2 Sections

A UPT file is made up of three distinct sections:

- the header line
- the variable definitions
- the file definitions

If the header line is missing, the file is classified as invalid and ignored.

In principle the variable definitions are optional, although it is just them which makes a UPT flexible and thus valuable.

The non-existence of any file definition, although syntactically safe, makes a UPT file absolutely useless. After all, a UPT is to generate files. If there is no file definition, a file simply cannot be created.

4.3 Section Order

The sections of a UPT file must appear in the following order:

- 1st: the header line
- 2nd: the variable definitions - if there aren't any: the file definitions
- 3rd: the file definitions (if there are variable definitions)

If the header line is not at the beginning of the file - apart from comments or any white space -, the file is classified as invalid and ignored. The header line is terminated by a semicolon.

The variable definition section is terminated by the beginning of the file definition section which starts with (the first) "@@". Variable definitions which follow the file definition(s) are considered simple text, belonging to the (last) file.

The file definition section is terminated by the end of the UPT file.

4.4 Size

The size of the UPT file and such the number and the size(s) of the generated file(s) are virtually unlimited.

5. Syntax

5.1 Comments

You can write standard C++ comments ("/* ... */ and "//") as usual. Inside of the variable definition section they behave as expected. Inside of the file definition section they become comments in the generated file. Putting "/*" into the variable definition section and "*/" into the file definition section leads to weird results.

5.2 White Space

Inside of the header line and the variable definition section, but outside of a word or a number, extra white space doesn't count.

Inside of the file definition section and starting with the second line of each file definition the same rule applies inside of expressions. Outside of expressions any white space defines the layout of the respective file. That simply means: white space appears as white space in the generated file.

5.3 Basic Rule

Inside of the file definition section any text which is not part of an expression appears unchanged in the generated file.

5.4 Delimiters

5.4.1 "@@"

"@" marks the beginning of a file definition. Everything up to this delimiter is part of the header line, a variable definition, or a previous file definition.

But "@@" is a valid delimiter only at the beginning of a line. In this case even white space matters. If "@@" are not the very first two characters of the line, they represent just plain text.

"@" functions not only as a delimiter but mainly as (the beginning of) an expression.

5.4.2 "??"

Although "?" marks the possible nullification of a file definition, it is more an expression than a delimiter.

It must appear inside of a file definition. If not, the UPT file is classified as invalid and ignored.

"" is a valid delimiter only at the beginning of a line. In this case even white space matters. If "" are not the very first two characters of the line, they represent just plain text.

"" makes sense only in cooperation with an expression. Standing alone it makes its line disappear.

5.5 Expressions

5.5.1 Basic Rule

As the UPT file is evaluated by CParser, you can use any valid numerical or logical C expression or any combination of them, wherever an expression is permitted. The numeric values are of type "double".

Expressions are meant to be used inside of the file definition section.

You can also use such an expression inside of the variable definition section as a default value for an option variable and a select variable, if this makes sense to you.

NEVER USE AN EXPRESSION TO SET THE DEFAULT VALUE OF A FILENAME VARIABLE OR AN ID VARIABLE! YOUR COMPUTER IS LIKELY TO CRASH AS SOON AS YOUR TEMPLATE IS SELECTED.

5.5.2 "@@filename_expression"

"" marks the beginning of a file name. The name starts immediately after the second "@" (with the third character of the line) and ends with the last character of the logical line. If you enter a filename as a literal, make sure that it is valid, especially if you use white space. The validity of a filename depends on the OS.

There are three typical patterns (the file extensions are just examples), although other or more complicated expressions are possible:

- @@<:PACKAGE:>.cpp
- @@<:filename_variable:>.hpp
- @@filename_literal.txt

5.5.3 ""expression"

"" decides whether a file is to be created. It evaluates the following expression. If the result is true, the file is created. If the result is false, the entire file is discarded. This holds, even if there are lines above the line containing the "".

"" has a second effect. Beginning with the first file template containing a ""expression" ("" alone does not suffice!) "<:PACKAGE:>" retains the path - if any - in the variable. Without such a line the path - if any - is removed from the variable. That means:

Independent of the value of "expression" the filename templates of all the files preceding the file template containing the first ""expression" don't show the path - if any - in <: PACKAGE :>.

If ""expression" evaluates to "true" "<:PACKAGE:>" in the containing file template shows path - if any - and filename.

If ""expression" evaluates to "false" the containing file template simply doesn't show up.

Independent of the value of "*expression*" the filename templates of all the files following the file template containing the first "*??expression*" DO show the path - if any - in <: PACKAGE :>.

5.5.4 "<: ... :>"

"<:" and ">" convert the enclosed literal into an expression which is evaluated. The result is then displayed without the delimiters.

A string, enclosed in quotation marks, yields exactly this string. Without quotations marks it is considered a variable. If this variable has been defined, its value is displayed. Otherwise, nothing is shown.

You can combine numerical expressions and strings, e.g.:

```
<:condition_1 ? "text_1" : "text_2":>
```

You cannot nest "<: ... :>".

"<: :>" and a single "<:" show a corresponding error message. A single ">" is just a plain string.

5.5.5 "<:? *expression* :> ... <:..:>"

This pattern emulates "if (*expression*) ...;"

White space inside of the "<:..:>"s doesn't matter. Between them it does: it produces white space in the resulting file.

5.5.6 "<:? *expression* :> ... <:/:> ... <:..:>"

This pattern emulates "if (*expression*) ...; else ...;"

White space inside of the "<:..:>"s doesn't matter. Between them it does: it produces white space in the resulting file.

5.6 "PACKAGE"

5.6.1 Description

"PACKAGE" is the predefined identifier of a variable to which the contents of the "Package name" input field are assigned.

5.6.2 Usage

The input is filtered. Only alphanumeric characters, "_", "/", and "\" are accepted. That means that you can enter a path but not a drive specification as ":" is not a valid character (e. g., Windows). "\" is immediately transformed into "/".

Whether the path in "PACKAGE" - if any - is shown by <: PACKAGE :> depends on the preceding existence of a "*??expression*" in the UPT file. Refer to section 5.5.3 "*??expression*".

```
... <: PACKAGE :> ...
```

5.7 "PACKAGE_UPPERCASE"

5.7.1 Description

"PACKAGE_UPPERCASE" is the predefined identifier of a variable to which the filename part, but not the path part - if any -, of the "Package name" input field, transformed into uppercase letters, is assigned. If there were uppercase letters before, they are not changed.

The reason for the existence of this variable is a well-established practice to write uppercase include guards in .hpp- (.h-) files.

"PACKAGE_UPPERCASE" is not available with Ultimate++ releases prior to "609-dev3".

5.7.2 Usage

"PACKAGE_UPPERCASE" never shows the path - if any - in the "Package name" input field.

```
... <: PACKAGE_UPPERCASE :> ...
```

5.8 Header Line

5.8.1 Description

The header line identifies a UPT file as such, names the template and categorizes it.

5.8.2 "template"

The "template" keyword identifies the UPT file as such. If this keyword is not at the beginning of the file - apart from comments or any white space -, the file is classified as invalid and ignored. The keyword must not be enclosed in quotation marks.

5.8.3 "*Name*"

This is the name of the UPT file which appears in the template list of the "Create new package" dialog. It must be enclosed in quotation marks. If the name is missing, the file is classified as invalid and ignored.

5.8.4 "main", "sub"

These keywords which must not be enclosed in quotation marks, determine the scope of the UPT file.

The "Select main package" window contains the option "All packages". If this option is checked, all the "main" UPT files only are presented in the template list of the "Create new package" dialog. If this option is unchecked all the "sub" UPT files only are shown in the template list of the "Create new package" dialog.

A UPT file can be both "main" and "sub".

If neither keyword is used, the UPT file is silently ignored.

5.8.5 Usage

The header line is terminated by a semicolon.

```
template "name" [main | sub | main sub | sub main];
```

5.9 "filename"

5.9.1 Description

"filename" defines a variable which is determined to host a filename. Using it makes a named edit field appear. The input is filtered. Only alphanumeric characters, "_", and "." are accepted. That means that you cannot enter a path or a drive specification (e. g., Windows).

5.9.2 Definition

The keyword and the variable name must not be enclosed in quotation marks, whereas the title must. Doing otherwise results in an error message and the UPT file is ignored. You can set a literal, enclosed in quotation marks, as default. Using a variable name instead, although syntactically correct, doesn't work. The variable definition is terminated by a semicolon.

```
filename "title" variable_name [= "default"];
```

5.9.3 Usage

```
... <: variable_name :> ...
```

You can compare filename variables and act based on the result of this comparison.

5.10 "id"

5.10.1 Description

"id" defines a variable which is determined to host a string. Using it makes a named edit field appear. The input is filtered. Only alphanumeric characters and "_" are accepted.

5.10.2 Definition

The keyword and the variable name must not be enclosed in quotation marks, whereas the title must. Doing otherwise results in an error message and the UPT file is ignored. You can set a literal, enclosed in quotation marks, as default. Using a variable name instead, although syntactically correct, doesn't work. The variable definition is terminated by a semicolon.

```
id "title" variable_name [= "default"];
```

5.10.3 Usage

```
... <: variable_name :> ...
```

You can compare id variables and act based on the result of this comparison.

5.11 "option"

5.11.1 Description

"option" defines a variable which is determined to host a boolean value. Using it makes a named option box appear.

5.11.2 Definition

The keyword and the variable name must not be enclosed in quotation marks, whereas the title must. Doing otherwise results in an error message and the UPT file is ignored. You can set either "0" or "1", enclosed in quotation marks or not, as default. Everything else is evaluated as "false". Using a variable name instead, although syntactically correct, doesn't work. The variable definition is terminated by a semicolon.

```
option "title" variable_name [= [""]0 | 1[""]];
```


5.11.3 Usage

```
... <: variable_name ? number_1 : number_2 :> ...
... <: variable_name ? "string_1" : "string_2" :> ...
... <: ? variable_name :>number<: . :> ...
... <: ? variable_name :>string<: . :> ...
... <: ? variable_name :>number_1<: / :>number_2<: . :> ...
... <: ? variable_name :>string_1<: / :>string_2<: . :> ...
```

You can compare option variables and act based on the result of this comparison.

5.12 "select"

5.12.1 Description

"select" defines a variable which works like an enumeration. Integer values, starting with 0, are assigned to named options. Using "select" makes a named DropList appear, of which the options can be selected but not edited.

5.12.2 Definition

The options must be listed inside of parentheses. The keyword and the variable name must not be enclosed in quotation marks, whereas the options and the title must. Doing otherwise or giving no option at all, results in an error message and the UPT file is ignored. The options must be separated by commas. If not, they are concatenated. You can set any value, enclosed in quotation marks or not, as default, but only integer values in the range "0 to number_of_last_option - 1" are accepted. Everything else makes no default option appear. The same holds for variables, be they defined or not. If you give no default at all, the first option becomes the default. The variable definition is terminated by a semicolon.

```
option ("option_1", "option_2", ...) "title" variable_name
    [= 0 | ... | number_of_last_option - 1];
```

5.12.3 Usage

```
... <: variable_name == zero_based_option_index :> ... <: . :>
... <: variable_name > zero_based_option_index :> ... <: . :>
... <: variable_name >= zero_based_option_index :> ... <: . :>
... <: variable_name < zero_based_option_index :> ... <: . :>
... <: variable_name <= zero_based_option_index :> ... <: . :>
... <: variable_name != zero_based_option_index :> ... <: . :>
```

You can select filename variables and act based on the result of this comparison.

5.13 Undefined variables

Using undefined variables in syntactically correct places causes no harm. Nevertheless an undefined variable has no value, even not 0.

6. Template Dialog Layout

The "Create in"-DropList always shows the currently selected assembly as well as the "uppsrc" assembly.

The templates are sorted alphabetically. If you have many templates, the template list will get scrollbars. So the number of your templates is virtually unlimited.

The variables appear in the order as entered. If you have many variables, they may not all be shown.

Just enlarge the window. But this dialog section will not get scrollbars, so the number of variables is indeed limited.

The preview window displays TAB characters as small upright rectangles, but after opening the generated project files these characters show up as usual.

7. Deployment

TheIDE looks in the following directories for UPT files:

- *selected assembly*
- *upp*
- *uppsrc*

Subdirectories are not scanned.

8. Epilogue

This documentation is valid for Ultimate++ releases beginning with "609-dev3" (September 24, 2006).

If you find any mistakes, be they just linguistic or typographic, please report them on the Ultimate++ forum (<http://www.arilect.com/upp/forum/>).

If something is not clear or seems to be not clear, please report so on the Ultimate++ forum.

If this documentation is insufficient or seems to be insufficient, please say so on the Ultimate++ forum.

Appendix: UPP File Syntax

Each package has a package definition file, the name of which must be the same as the name of the package folder. The extension is ".upp".

If you create Ultimate++ GUI applications, the UPP file normally has the following three sections:

```
uses
    CtrlLib;

file
    file_1.ext,
    file_2-ext,
    ...;

mainconfig
    "" = "GUI flag_2 ...";
```