Unimal 2.0b

Application note 7

Guarding the include files

Documentation revision 2.0b

Techniques:
When to guard and when not to guard Unimal include files
Using uAutoLine as an include file guardian

MacroExpressions
http://www.macroexpressions.com

Table of contents

Foreword

In C-like languages where information is shared among translation units via include (header) files, a common idiom is to guard an include file, say, `header.h`, like so:

```
#ifndef HEADER_H_
#define HEADER_H_
.........
Real content
.........
#endif
```

The net effect of the `ifndef/define/endif` guard is that however many times, directly or indirectly, the header is included, only the first inclusion has any effect.

In this Application Note, we'll consider when a similar technique is useful in Unimal and how to implement it easily when needed.

To guard or not to guard?

In C, guarding header files is common but not universal. For instance, the C system header assert.h is (typically) not guarded, and that's on purpose.

Taking this direction in Unimal: When is it reasonable to guard an include file and when is it not?

Consider, for the example, a table of definitions like

```
#MP Expand Something(1)
#MP Expand Something(10)
#MP Expand Something(11)
#MP Expand Something(110)
```

If it is placed in an include file, we probably want it to be included as many times as we include the file, considering that the macro `Something` may by design expand differently depending on some controlling parameter. In this case, we don't want to guard the include file.

A macro definition placed in an include file does not require a guard: Unlike C, Unimal detects that it's the same textual definition and ignores it automatically. However, a guard does no harm, and multiple inclusions of an include file with many macro definitions is likely to be processed faster if guard is present.

A somewhat odd example is a conditional definition of a macro, like

```
#MP If condition
```

```
#MP    Macro Foo
::: In Foo :::
#MP    Endm
#MP Else
#MP    Macro Foo
::: In Alternative Foo :::
#MP    Endm
#MP Endif
```

If `condition` changes between two inclusions of an include file with such a construct, the macro definition will be picked up from a different block and a macro redefinition error will result. So, if the intention is to get the first-seen definition of `Foo`, we need a guard. On the other hand, this construct may be used to trap an unexpected change of `condition` and break the build if it happens. In this case, a guard is not wanted.

A general conclusion is that it is up to the programmer to decide whether to guard a Unimal include file, but the need in guarding is far less than it is in C.

A C-style guarding

It is straightforward to use a C technique of guarding: For `myfile.u`, we can write

```
#MP If !Defined(MYFILE_U_)
#MP MYFILE_U_ = 0 ;any value, just to define a parameter
.........
Real content
.........
#MP Endif
```

Arguably there is a weakness: There is no correlation between the file name and the guardian name; this also applies to C/C++. Accidentally, a name of a guardian can be used in different include files as a result of file renaming and/or cut-n-paste mistakes. People learned to live with this, but Unimal offers some automation which eliminates the weakness, as described below.

Using uAutoLine for guardian

Recall that the *string* value of the macro parameter `uAutoLine` is the name of the current file (by default, as seen by Unimal, or if the –p command-line switch is used, a fully qualified name).
The Unimal composite names mechanism allows to use `%suAutoLine` as the name of a macro parameter; for an include file `myfile.u` the name becomes `myfile.u` and cannot be used literally. If we assign a value to a parameter with this name, it is very likely to be unique:

```
#MP If !Defined(%suAutoLine)
#MP %suAutoLine = 0 ;any value, just to define a parameter
.........
Real content
.........
#MP Endif
```

The first two lines *are* good for thoughtless cutting and pasting; the match of the file name and the guardian name is automatically ensured. [Oh yes, it is possible to contrive a name

by hand to confuse this scheme. But it would be a result of a conscious effort, not of an accidental mistake.]