マクロ~発展編

By Jonathan Le, Technical Support, 4D, Inc.
Technical Note 05-01

（原題: Advanced Method Editor Macros）

概要

単なるメソッドの入力支援にはとどまらないマクロの奥深い活用方法を紹介する。

マクロとは？

一定のルールに従ってコードを自動生成する機能である。
たとえば、4Dに標準で附属している"If"マクロは、自動的に以下のコードを生成する。

```
If()
End If
```

これ自体ではコードとして未完成だが、多少は手間が省ける。

バージョン2003では、前述の"If"に代表されるように、コードの定型部分を事前に入力するのが主な用法だった。タイピングに要する時間を少しでも節約するとともに、不用意なタイプミスを防ぐ意味があった。さらにバージョン2004では、METHODタグによって、マクロでメソッドコールをすることが可能になり、機能性が飛躍的に向上した。

マクロの基本

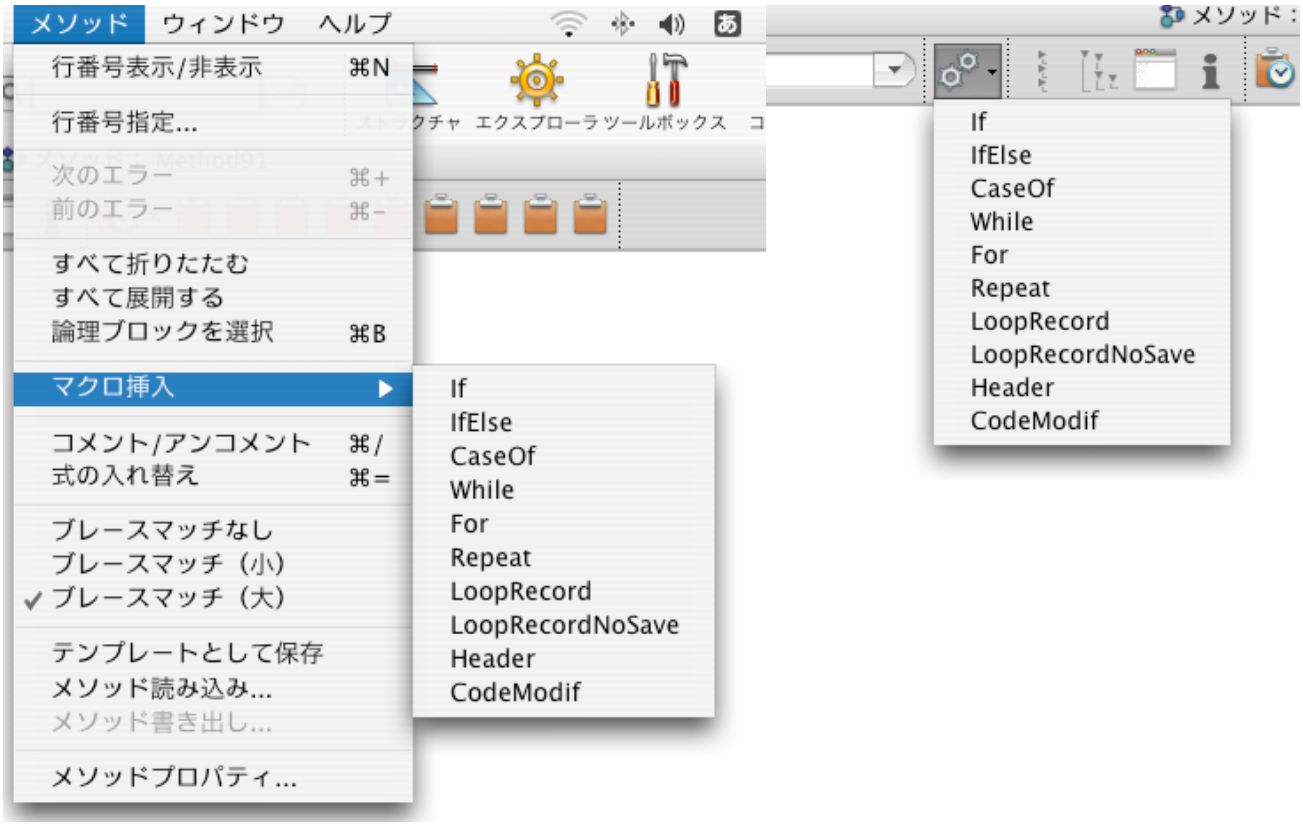
マクロファイルはアクティブな4Dファイル内の専用フォルダに設置する。

マクロタグ

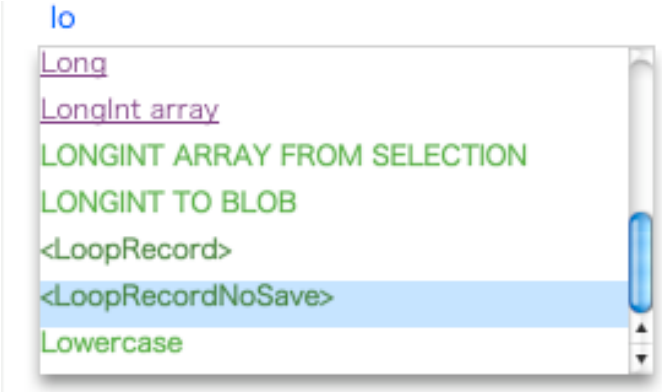
マクロの定義は、4D独自のマクロタグを使用して記述し、XML形式で保存される。

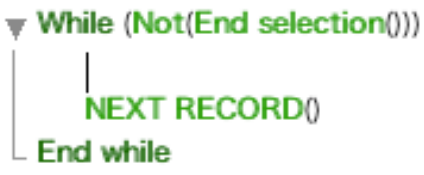
マクロを入力する方法

メソッドエディタのマクロアイコンをクリックするか、メソッドメニュー/マクロ挿入を選択する。



タイプアヘッド（予測入力）でマクロを入力することもできる。マクロ冒頭の数文字をタイプし、Tabキーを押すと候補が表示され、マクロを選択して再度Tabキーを押すとマクロが実行（挿入）される。





マクロファイルの場所

アクティブな4Dフォルダの場所は、通常、WindowsではApplication Data、MacOSでは
Application Supportフォルダ内に設置されている。

確実にその場所を知るには、以下のコマンドが便利である。

```
C_TEXT($active4D)
$active4D:=Get 4D folder
```

4Dは自動的に“Macros.xml”という標準のマクロファイルを作成する。
同じ階層の"Macros"フォルダに、任意でXML形式のマクロファイルを追加することもできる。

マクロを作成する

典型的なマクロは、以下のような記述で定義されている。

```
<macro name=”HelloWorldMacro”>
<text>
Hello, World!
</text>
</macro>
```

この場合、マクロ”HelloWorldMacro”を実行するとメソッドエディタには“Hello, World!”という文字列が入力される。実際の4Dコードを使用するとすれば、以下のようになる。

```
<macro name=”HelloWorldMacro”>
<text>
ALERT(“Hello, World!”)
</text>
</macro>
```

このように<text>タグに囲まれた文字列は、メソッドエディタに入力される。
使用できるタグの一覧は次のとおりである。

タグ	説明
macros	マクロファイルの冒頭と末尾に一度ずつ使用し、マクロファイルであることを宣言するタグ。
macro	マクロを定義するタグ。属性nameでマクロの名前を定義する。（必須）
caret	マクロ実行後のカーソル位置を指定するタグ。
selection	マクロ実行時に、選択されていた文字列と置き換える内容を指定するタグ。
use_os	カレントマシンのログイン名と置換されるタグ。
date	システム日付と置換されるタグ。属性formatでフォーマットを指定する。
time	システム時間と置換されるタグ。属性formatでフォーマットを指定する。
method_name	マクロを呼び出したメソッド名と置換されるタグ。
method	プロジェクトメソッドを実行するタグ。

たとえば、4Dに標準で附属のマクロ"Header"は以下のようなコードで定義されている。

```
<macros>
<macro name="Header">
<text>
` ------------------------------------------------
` User name (OS):
<user_os/>
` Date and time:
<date format="0"/>
,
<time format="0"/>
` ------------------------------------------------
` Method:
<method_name/>
` Description
`
<caret/>
`
` Parameters
` ------------------------------------------------
</text>
</macro>
</macros>
```

マクロ"Header"は、標準的なメソッドヘッダを生成するために使用されるマクロである。メソッド名、作成者、タイムスタンプがコメントとして出力され、説明を記述する行にカーソルが移動する。

応用編

"If"マクロをもう少し拡張してみよう。

```
<macros>
<macro name="IfWrap">
<text>
If (<caret/>)
<selection/>
 End If
</text>
</macro>
</macros>
```

オリジナルと異なるのは、IfとEndifの間に<selection/>タグで事前に選択されていた文字列が挿入される点である。

仮に次のコマンドを選択した状態で、マクロ"IfWrap"を使用したとする。

```
ALL RECORDS([My  Table])

結果は次のようになる。

If ()
ALL RECORDS([My  Table])
End if
```

カーソルは()の間に移動し、すぐに条件の記述ができる体制になる。

```
If (Records in table([Table  1]))
ALL RECORDS([My  Table])
End if
```

METHODタグの使用

はじめに、次のメソッドGetTheTimeを用意する。

```
`  Take  one string as input
C_TEXT($1;$string)
$string:=$1

`  Display  the time and echo the input in an alert
ALERT("The  time is "+String(Current  time;5)+".  I say, \""+$string+".\"")
```

次に、このメソッドをコールするマクロを作成する。

```
<macro name="GetTheTime">
<text>
<method>
GetTheTime("<method_name/>")
</method>
</text>
</macro>
```

メソッドには引数として任意の文字列を渡すことができるが、ここではメソッド名を文字列として渡すために引数に<method_name>タグを使用した。<text>タグの間にメソッドをコールするタグを挿入しているので、結果はメソッドエディタに出力される。

特殊なプロセス変数

マクロと合わせて使用することのできる以下のプロセスが追加された。

INPUT

変数名	タイプ	説明
_textSel	テキスト	選択されたテキスト（32000文字まで。超えると""）
_blobSel	BLOB	選択されたテキスト
_selLen	倍長整数	選択されたテキストの長さ
_textMethod	テキスト	メソッド本文（32000文字まで。超えると""）
_blobMethod	BLOB	メソッド本文
_methodLen	倍長整数	メソッド本文の長さ

OUTPUT

変数名	タイプ	説明
_textReplace	テキスト	返り値、デフォルトは""
_blobReplace	BLOB	返り値
_action	倍長整数	返り値の処理方法 0=なし 1=_textReplaceを挿入 2=_blobReplaceを挿入 3=textReplaceで置換 4=blobReplaceで置換

以上の変数は、マクロで呼び出されたメソッドで使用することができる。

使用方法

デモDBのマクロ“ReverseIt”では、メソッドエディタで選択されている文字列をプロセス変数_textSelで受け取っている。<selection/>タグでも同様のことができるが、その場合は受け取った値に対して何の処理もできない点が異なる。

メソッドエディタで文字列をハイライトしてマクロを使用すると、文字列が逆になる。

マクロによって呼び出されているのは次のようなメソッドである。

```
` used by the macro in 4D
C_TEXT(_textSel)
C_TEXT(_textReplace)
C_LONGINT(_action)

` locals
C_LONGINT($i)

` check the length of what the user selected at the time of the macro call
If (Length(_textSel)>0)
_textReplace:="" ` reverse it
For ($i;0;Length(_textSel)-1)
_textReplace:=_textReplace+_textSel[[Length(_textSel)-$i]]
End for

` specify that we want to insert
text _action:=1
Else

` do nothing otherwise
_action:=0
End if
```

ローカル変数だけでなく、マクロで使用する変数もメソッドでコンパイラ宣言している点に注目。

開発上の注意点

マクロが呼び出したメソッドは、同一プロセスで実行される。デザインモードは
メソッドの終了待ち状態になるので、よほどの短いメソッドでない限り、マクロが呼び出すメソッドで新規プロセスを立ち上げるのが望ましい。

BLOBかテキストか

テキストの文字制限を考慮し、確実に32000文字以下であると分かる場合以外はBLOBを使用するのが望ましい。

ダイアログの利用

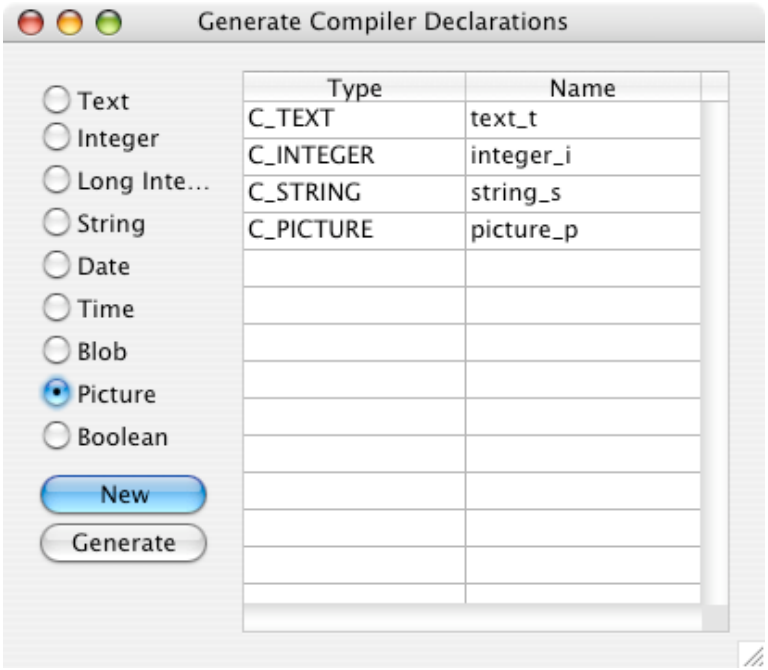
マクロが呼び出すメソッドではどのコマンドでも使用できるので、例えばウィザードのような入力フォームを開くマクロを作成することができる。（例: マクロVariableWizard）

このマクロを呼び出すと、ダイアログが表示され、コンパイラ宣言をGUIでまとめて作成することができる。

コードは以下のとおりである。

```
C_BLOB(_blobReplace)
C_LONGINT(_action)
C_LONGINT($windowRef)
C_LONGINT($i)
ARRAY  TEXT(ParameterTypes;0)
ARRAY  TEXT(ParameterNames;0)

$window:=Open form window([Macros  Example];"VariableWizard";Plain fixed size
window ;Horizontally Centered ;Vertically Centered )
DIALOG([Macros  Example];"VariableWizard")
CLOSE  WINDOW($window)

If (OK=1)  `_action  of 2 means insert into the method editor
_action:=2  `  Here  we just loop through our array and generate the code

For ($i;1;Size of array(ParameterTypes))
$variableType:=ParameterTypes{$i}  `  the variable type
$variableName:=ParameterNames{$i}  `  the variable's name

`  here we write the declaration
$declaration:=$variableType+"("+$variableName+")"+Char(Carriage return )

`  Finally,  append that bit of text into _blobReplace variable
TEXT TO BLOB($declaration;_blobReplace;Text  without length ;*)
End for
End if
```

メソッドは、フォーム[Macros Example];"VariableWizard"を開き、取得した変数タイプと変数名の配列に基づいてコンパイラ宣言のテキストBLOBを作成する。




```
C_TEXT(text_t)
C_INTEGER(integer_i)
C_STRING(string_s)
C_PICTURE(picture_p)
```

追記

デモDBには、例題マクロの他に、Macro Packコンポーネントがインストルされている。Macro Packには、紹介したマクロの他に多数の機能的なマクロが用意されている。

デモDBでメソッド“MEM_InstallMacros”を実行すると、Macro Packコンポーネントのマクロがアクティブな4Dフォルダにコピーされるので、以降、4DでMacro Packのマクロが使用できるようになる。

Macro Packのコンポーネント、ソースおよびドキュメントは、http://www.4d.comで。