

# SHORT PML GUIDE

*By Adnan Dervišević*

## INTRODUCTION

PML (Programmable Macro Language) is an important aspect of PDMS. It can make life easier for designers, drafters and administrators by automating some common tedious tasks, reviewing, etc. The goal of this guide is to cover some basic principles of programming through an example macro that fixes orientation of equipment in design module. For more information, I suggest reading the *Software Customization Guide* and *Software Customization Reference Manual*, both supplied with PDMS.

At this moment, PML is in its second revision called PML2 which is an object oriented language (well almost, but that is beyond the scope of this guide). It is an extension of original PML1 specifically designed to handle forms & menus. Macro file that accompanies this guide mostly uses PML1 syntax, as it is a good starting point for further learning.

## GETTING STARTED

All you need to write PML code is a simple text editor. Notepad will do. However, I strongly recommend using editor with syntax highlighting. Crimson Editor (now developed by Emerald Editor community) is a good choice. You can download the latest version from <http://sourceforge.net/projects/emeraldeditor/files/crimsoneditor/crimson-editor-2.72-r286/cedt-286-setup.exe/download>

After installation completion, set it up to recognize PML syntax by choosing Tools/Preferences/File → Syntax Type

Just scroll down to *-Empty-* slot and enter:

Description: PML

Lang Spec: PDMS-PML.SPC

Keywords: PDMS-PML.KEY

## GENERAL PML FEATURES

Macro (.mac) is nothing more than a sequence of commands stored inside a single file. It is started from within PDMS command window by typing:

```
$M/%PATHNAME%\fixori.mac
```

For example, if we have fixori.mac stored inside folder TEST on drive C: syntax would be:

```
$M/C:\TEST\fixori.mac
```

Nowadays preferred way of using PML is through Functions (.pmlfnc), objects (.pmlobj) and Forms (.pmlfrm) but they won't be discussed here.

Variables in PML can be of any *built in* type:

- **STRING** – any text
- **REAL** – any numeric value (integer and non-integer)
- **ARRAY** – contains many values of the same type
- **BOOLEAN** – holds the values of logical expressions (TRUE or FALSE)

There are also *system-defined* and *user-defined* variable types.

All variable names beginning with '!' are LOCAL (they can be used only from within one function or a macro) and variables beginning with '!!' are GLOBAL which last until PDMS is closed. PML is **not** case sensitive so it doesn't matter if the letters are UPPER or lower case. Give all your variables meaningful names; it will pay up in the long run.

## MACRO FOR FIXING EQUIPMENT ORIENTATION

It is important that at the end of equipment design all EQUI and SUBE elements have an orientation of Y is N and Z is U. Usually, this is not the case as mistakes are often made with improper use of model editor. Following macro fixes that problem without changing the position of equipment.

-- FixOri v1.0 by Adnan Dervisevic

First few lines of the code are comments. Simple inline comment can be made by starting a line with -- or \$\*. Comments that spread over multiple lines are enclosed between \$( and \$).

```
$(
This is a
three line
comment
$)
```

## IF CONSTRUCT

General form of an **IF** construct is :

```
if (CONDITION is TRUE) then
  (command block)
elseif (CONDITION is TRUE) then
  (command block)
else
  (command block)
endif
```

As soon as one of the conditions is TRUE, following command block is executed and everything else after that is ignored. If none of the conditions are TRUE, then command block after **else** is executed. **Elseif** and **else** are optional.

Our **IF** construct checks if we are on correct level in hierarchy and displays an error if this is not the case:

```
If (!!ce.type neq 'EQUI') then
return error 1 'Please stand on EQUI level'
endif
```

## COLLECTIONS AND EVALUATING

```
var !subs collect all subequi for CE
```

Collect all reference numbers of sub equipments for current element (macro is started from an EQUI level) and store them in array called *subs*. Collection is a very useful way of creating an array of certain type of elements which satisfy certain criteria. Its syntax is as follows:

```
VAR !ArrayName COLLECT Class Selection criteria
```

**VAR** is a PML1 way of creating variables. It will return an ARRAY with references to subequipment as STRING. Array members are referred to as **!ArrayName[1]** , **!ArrayName[2]** ... etc

**Class** determines what types of elements are to be collected. In our case, it is ALL SUBEQUIPMENT.

**Selection criteria is optional** and it can be a logical expression, a physical volume (defined by the WITHIN keyword) or a hierarchy criteria (defined by FOR keyword - in our case only elements below current element are collected)

```
var !SubPos eval position in world for all from !Subs
```

Now we evaluate position for all elements from **!Subs** array .One new array is created, called **!SubPos** with positions of sub equipments relative to world. Read more about collections and evaluating in *Software customization guide* as they can be very useful.

## DO LOOPS

In PML there are no FOR, WHILE or REPEAT loops. They are all replaced by DO loop which has the following form:

```
do !variable from X to Y by Z
.
.
.
enddo
```

*!Variable, from, to* and *by* are optional. If *from* is not given, the counter starts from 1 by default.

In our macro, we will use the following loop:

```
do !x index !Subs
.
.
do !y index !Elements
.
.
enddo
enddo
```

This is the so-called *nested loop*. The inner loop goes through all the values of **!y** and then the outer loop moves to the next value of **!x**. You can see here that *from, to* and *by* have been replaced by *index*. It means that **!x/!y** will take the index values of arrays **!Subs/!Elements** from 1 to the maximum array index.

For example, if array **!Subs** goes from **!Subs[1]** to **!Subs[5]**, **!x** will take values from 1 to 5.

Sometimes it is more convenient to use *values* instead of *index*. That way **!x** would take values of each array member from first to last.

This nested loop has the following algorithm:

- Make subequipment with index **!x** current element in hierarchy.

```
!!CE = !Subs[!x].dbref()
```

**Dbref()** is a method that converts STRING type names of subs to DBREF type. This must be done, otherwise the PDMS will report an error. You can read about methods and objects in *Software Customization Reference Manual*.

- For each subequipment, collect all of it's elements and evaluate their positions and orientations relative to world.
- Store these positions and orientations in two dimensional arrays - **!PosMatrix** and **!OriMatrix**. This is what inner loop does.

*It is important to collect positions of subequipment and its elements before we do anything as these can sometimes change with orientation and we want the equipment to remain in the same position.*

Now we can start fixing the orientation:

```
EQUI  
Orientation Y is N and Z is U WRT/*
```

These two lines take us back to equipment and set its orientation to Y is N and Z is U, relative to world.

Second nested do loop is similar to previous one but instead of collecting, we are now setting the orientations and old positions to each sub and its members.

Interesting part here is the use of **\$** - escape character. Together with the character that follows it is treated as a special instruction to PML (escape sequence).

```
Position $!subpos[$!x] WRT/*  
Orientation $!OriMatrix[$!x][$!y] WRT/*  
Position $!PosMatrix[$!x][$!y] WRT/*
```

In case when variable is a part of some command or when you want to display it on a screen, you must put **\$** in front, otherwise PDMS will report an error. Some useful escape sequences are:

**\$P** outputs a message to a screen. **\$P Hello World!!**

**\$M** is used to run macros and db listings. **\$M/C:\TEST\fixori.mac**

**\$Q** after a command displays a list of all possible command attributes.

## CONCLUSION

This text covers only the very basics of PML. As I am still in the learning process of PML, errors are possible. Please send any comments and suggestions to my email: [the.adnan@gmail.com](mailto:the.adnan@gmail.com)