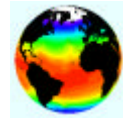




Climate Modeling and Global Change Team
CERFACS
The PALM group
24th January 2002



TR/CMGC/02/9

The Coding Conventions for the **Palm** MP

The teamwork for developing and maintaining software can greatly benefit from the adoption of an implementation standard that can make the code very easy to follow and to modify.

In addition to a well thought-out design of the algorithm and of the code architecture and to an exhaustive bench, a coding norm is required to fulfill this aim.

The purpose of a coding standard is to structure the files and the names of the associated functions and to produce a coding style that leads to well presented source code: such code should be well structured, well documented and thus easy to understand. In particular, a norm should provide standards for the recognition of the nature and of the scope of variables, partitioning of the code within sections, and a mean of extracting documentation from the source code.

The reference languages for the PALM software are Fortran 77, Fortran 90, C, Tcl and Shell and a particular relevance is given to the message passing aspects.

Such conventions will simplify the use of the ProDOC software that generates automatically the documentation associated to the code.

It has to be noticed that this new version of the coding norm for PALM had been simplified to make it easier to use.

1 Naming of the files and associated functions

Each file in the PALM MP software would gather all the functions used in a specific way. Thus, all the functions used by the driver will be found in the file driver.c or driver.f90, etc.

The names of the functions are defined using the following rule:

*Each function has a name made up of four components such as
(1,type)_(2,file)_(3,action)_(4,complement)*

These four components could be described as:

- the type: this component gives the type of the function. The first letter corresponds to the type of the variable the function returns, following the norm described after. The second letter is 'f' for function in C.
- the file: this component gives the domain in which the function will act. A file is containing functions that have the same nature.
- the verb: this component describes the action the function accomplishes.
- the complement: this component gives details about the action the verb described.

In C, the header files will follow the organization of the '.c' files. For each 'file.c' will correspond 'file.h'. In this last we will find:

- The declaration of the prototype of the functions that are described in 'file.c',
- The definition of the different structure used in the associated functions,
- The definition of the global variables.

Examples: Let's consider the actions done on the communications. This leads to consider the file 'comm.c'. This file will gather all the functions that are common for all of the actors of the software (driver, entities, etc...). If some functions are specific to the driver for the treatment of the communications, we will consider the file 'commdrv.c'.

In 'comm.h', we will find the definition of the structure and the global variables used in all the files that are related to the communications. We will also find the declaration of the prototypes of the functions that can be used by all the components of the software that interact with the communications.

The header file 'commdrv.h' will gather the same types of information except they are only useful for the driver part.

The function 'if_commdrv_allocate_table' will be declared in 'commdrv.h' and defined in 'commdrv.c'. This function will allocate the communication driver managed by the driver.

2 The extracting tool: ProDOC

In order to be able to get a complete documentation of each function, module or subroutine, in an easy and quick way, the ProDOC software had been implemented following the coding rules.

Four levels of extraction are available:

- the **overview level** that gives the title, the purpose and the interface of a function, a module or a program,
- the **external level** that gives in addition of the overview, the method, the externals, the files used for I/O, a reference to further documentation, and the author, the date and the modification for the function, module, or file.
- the **internal level** that gives in addition of the external level the structure of the code (i.e. the section titles, sub-section titles, etc...)
- the **global level** that gives all the coding lines.

ProDOC will recognize these levels reading the specific comment cards such as `‘/*****’` in C for the overview, `‘/****’` for the external, `‘/***’` for the internal and `‘/*’` for the global. The corresponding comment cards for the other languages will be obtained replacing the first cards by the ‘C’ for Fortran 77, ‘!’ for Fortran 90, or ‘#’ in Tcl or Shell.

3 Coding style

This part will detail all the coding conventions used to clarify the different parts of a code. It has to be noticed that each language has its own comments marks. The following examples will be described for the C language. The use of such conventions for other languages will just need the replacement of the comment card `‘/’` for the C by ‘C’ for the Fortran 77, ‘!’ for the Fortran 90 and ‘#’ for Tcl and SHELL. Finally, the comments in C will always need to end by `‘*/’` where the number of asterisk will correspond to the opening ones.

All of these conventions will be associated to the extraction levels they could belong to in ProDOC.

3.1 Initial comments

Each module, function or program should begin with a set of initial comments. These should contains:

- a *title*,
- the *purpose* of the module, function or program,
- the *interface details*, details output/input parameters, arguments, etc...
- the *method description*, principles of the algorithm,
- the *externals* or others routines called,
- the *files* used for I/O,

- a *reference* to further documentation,
- the *author* and *date* the routine was written, together with *modification details*.

The following table details for which level of extraction these comments will appear in the document generated by ProDOC:

| level of extraction | title | purpose | interface | method | externals | files I/O | reference | author... |
|---------------------|-------|---------|-----------|--------|-----------|-----------|-----------|-----------|
| overview | x | x | x | | | | | |
| internal | x | x | x | x | x | x | x | x |
| external | x | x | x | x | x | x | x | x |
| global | x | x | x | x | x | x | x | x |

3.2 Sections and separators

Each section should be separated from the previous one by a comment card followed by an asterisk ('/*' in C for example) and by the rest of the line of minus signs. No blank has to be put between the asterisk and the first minus sign.

Each section should begin with the comment expression '+/**', a plus and a title. Each subsection should begin with two plus, etc...

Each function, module or program should be separated from the previous one by a comment card followed by an asterisk and by the rest of the line of equal signs. No blank has to be put between the asterisk and the first minus sign.

3.3 Variables description, inclusion explanation, etc...

The most important variables should be described at their declaration. In Fortran 77 and TCI, the descriptions should be on separate lines following the declaration and triggered by 'C*V' or '#*V' starting in column1. The description must contain the name of the variable.

In Fortran 90 and in C, the description should follow the declaration on the same line and should be triggered by '!*V' or '/*V' respectively. The description must contain the name of the variable.

The same rule will apply for inclusion ('I' is used instead of 'V') and the use of modules ('U' is used instead of 'V').

3.4 Naming conventions

The purpose of the following naming convention is to convey, through prefix, the scope, and the nature of all variables within the program. This norm is compatible with the four languages for the common types of variables.

The structure of the name of a variable is **ns[a]_name** where n is a letter indicating the nature of the variable, s is a letter indicating the scope of the variable, a

is an optional letter indicating if the variable is an array or not, and name is a mnemonic name that is not exceeding 10 letters. The convention for n and s is detailed in the following table:

| | INT | REAL | DOUBLE | LOGICAL | CHAR | TYPE | FILE | VOID | MPI |
|-----------|-------|-------|--------|---------|-----------|--------|------|------|-----|
| | int | float | double | | character | struct | file | void | mpi |
| Global | ig_ | rg_ | dg_ | lg_ | cg_ | sg_ | fg_ | vg_ | mg_ |
| Local | il_ | rl_ | dl_ | ll_ | cl_ | sl_ | fl_ | vl_ | ml_ |
| Dummy | id_ | rd_ | dd_ | ld_ | cd_ | sd_ | fd_ | vd_ | md_ |
| Parameter | IP_ | RP_ | DP_ | LP_ | CP_ | SP_ | FP_ | VP_ | MP_ |
| Function | if_ | rf_ | df_ | lf_ | cf | sf_ | ff_ | vf | mf_ |
| Loop | ib[_] | | | | | | | | |

The loop index `ib` will be enough for one-level loops. In case of nested loops, indexes should be named `ib_name` where `name` is a mnemonic key.

The parameters should be written in capital letters. The constants, only used by the PALM user, should begin with `PL_` whereas the others will follow the rule define just before. All of those constants should be defined in header files (.h) with the pre-compilation instruction `'#DEFINE'`.

3.5 Comments in Latex

The user should be able to write some comments using latex instructions. The latex instructions should begin with the instruction ‘beginlatex’ and end with the instruction ‘endlatex’.

It is also possible to directly use ‘beginitemize’ and ‘enditemize’ with ‘\item’ to generate a list of items. In order for ProDOC to be able to generate the dependence tree of the functions and files, it is important not to use the latex functionality in the ‘externals’ category.

4 Examples

The following part is an illustration of the coding rules in a code, and the resulted documentation of ProDOC on such file.

```

/**** File exam.c – File that gathers an example of functions following
        the coding norm.

    Purpose:
        This module is the illustration of the coding norm.
        This first part describes the general aim of the file.
                                                                    *****/
/***/
History:
-----
Version   Programmer   Date       Description
-----
    1.0    Mister PALM   08/01/2002  Creation
                                                                    ***/
/*-----*/

/** + Declarations **/

/** ++ Include files **/

#include <include_file.h> /*I first include file */

/** ++ No global declarations **/

/*=====*/
```

```

int if_exam_show_first(const void *cda_first, const void *cda_second) {

    /*** if_exam_show_first – First example of the file

    Purpose:
        Describe a first example

    Interface:
        if_exam_show_first (cla_first, cla_second)
                                                                *****/

    /***
    Method:
        beginitemize
            \item Call a first routine,
            \item Call a second routine.
        enditemize

    History:
    -----
    Version   Programmer   Date       Description
    -----
        1.0     Mister PALM    08/01/2002  Creation
                                                                *****/

    /*-----*/

    /** + Declarations **/

    /** ++ Local variable **/

    int il_res ; /*V il_res is a local integer */

    /*-----*/

    /** + First treatment **/

    /** ++ First instruction of the treatment **/
        il_res = memcmp(cda_first , cda_second, 1*sizeof(int)) ;

    /** ++ Second instruction of the treatment **/
        il_res = il_res-1;

    /*-----*/

    /** + Second treatment **/

```

```
/** ++ only one instruction in the second treatment **/
```

```
il_res = il_res * 10;
```

```
/*-----*/
```

```
/** + Return result **/
```

```
return il_res;
```

```
}
```