The Preliminary Guides to the MegaWave2 Software, Versions 2.x

Volume One

MegaWave2 User's Guide

by Jacques Froment

Ecole Normale Supérieure de Cachan
61 avenue du Président Wilson
94235 Cachan cedex, France

May 5, 2004

`http://www.cmla.ens-cachan.fr/Cmla/Megawave`

Contents

1 Introduction

1.1 What is MegaWave2

MegaWave2 is a software environment designed to help people to write algorithms on signal processing and image analysis. The user is assumed to be a non-computer scientist, in particular he is likely to be a mathematician (notice however that a good knowledge of the C language is assumed).

This means that the programmer should focus his mind on the algorithm only and not on pure computer problems (such as how to read an image, how to display a movie on the screen, how to select a curve with the mouse, how to manage an object library,...)

The solution adopted by MegaWave2 is the concept of modules, that is, of black boxes where the algorithms are described as C functions, without any assumption about the context in which the modules may be executed. It is the task of the MegaWave2 compiler (basically a MegaWave2 preprocessor followed by a standard C compiler) to make from a module a run-time command, a function of a library or a function to be run under a window-oriented interpreter.

1.2 Motivations and history

MegaWave2 has been basically created to help researchers in writing algorithms in the domain of signal processing and image analysis. MegaWave2 allows a diffusion of the scientific information along all the members of our research team (and between members of associated teams also): a module written to experiment a process may be used by somebody else several months later. MegaWave2 was born in 1993 at the CEREMADE, a mathematical laboratory of the Paris-IX Dauphine University, France. It succeeded to MegaWave (renamed MegaWave1) which essentially was a set of C files related to the wavelet transform (actually, MegaWave1 was a quite different system since it did not have any specific pre-processor.The advantages of MegaWave1 were the simplicity and the documentation, which was written in french and therefore which did not misuse the Shakespeare language. Strong protests from our U.S. partners have urged us to change that !). Notice that in MegaWave2, if the wavelet part still exists, it only represents a small piece of all the mathematical methods for image analysis that can be found. Since 1999, MegaWave2 is no longer directed by the CEREMADE but by the CMLA (Centre de Mathématiques et de Leurs Applications), CNRS UMR-8536, at the Ecole Normale Supérieure de Cachan, Cachan, France.

We decided to put MegaWave2 in the public domain in 1994 (under some restrictions with version 1.x), since some algorithms may be worthy of interest for people working in the field of signal processing and image analysis. The source of each algorithm is given, and therefore each researcher in the world can check it and understand it more deeply than he could do by reading the published article. Thus, we experiment a new kind of scientific publication: the "soft-publication". The main scientific interest of soft-publication in image processing is to offer reproducible research.

Moreover, we put together with the modules the whole MegaWave2 environment, so people can write new modules which may use (or not) our algorithms. Thus we hope to help the scientific

community, by encouraging (in the limit of our capacities) the communication and the emergence of new ideas.

Please notice that, because we could not check MegaWave2 on all possible computer architectures and systems, there is only a limited number of computers on which you may run the software without encountering difficulties. But even if your computer is not compatible with the ones we have, you still can read the algorithms by reading the sources of the modules and even better, since MegaWave2 version 2.x, you may adapt the system so that it can run on your computer.

Last, please be indulgent with this software (which is not perfect - far from that - and which is still in development). It has been written essentially by mathematicians (and not by computer scientists), using the short time left after teaching and research activities.

1.3 Authors and acknowledgements

As MegaWave1, MegaWave2 has been created by *Jacques Froment*, but with the help of a greater number of people. Special thanks to *Sylvain Parrino* who helped me to write the preprocessor, and to *Claire* and *Lionel Moisan* who revised the text of the guides. New developments in MegaWave2 are still directed by *Jacques Froment*, with the help since 1998 of *Lionel Moisan* who is in charge of collecting and updating public modules and user's macros. It is hardly possible to cite here all the people who participated in MegaWave2 by writing new modules, or simply by giving me valuable advice. Here is the list of people who wrote modules included in the public MegaWave2 package Version 2.30, released in May 2004 :

Andrés Almansa, Chiaa Babya, Toni Buades, Frédéric Cao, Vicent Caselles, Antonin Chambolle, Guillaume Charpiat, Thierry Cohignac, Bartomeu Coll, Jean-Pierre D'Alès, Françoise Dibos, Vincent Feuvrier, Jacques Froment, Frédéric Guichard, Yann Guyonvarc'h, Yan Jinhai, Claire Jonchery, Georges Koepfler, Saïd Ladjal, Kamal Lakhiari, José-Luis Lisani, Simon Masnou, Lionel Moisan, Pascal Monasse, Pablo Muse, Denis Pasquignon, Florent Ranchin, Amandine Robin, Olivia Sanchez, Catalina Sbert, Frédéric Sur.

1.4 Main changes between MegaWave2 versions 1.x and versions 2.x

This guide is intended to be the User's Guide of MegaWave2 versions 2.x and it should not be used with former versions 1.x (check the version number with `mwvers`). For those of you who were familiar with MegaWave2 versions 1.x, I give below a short list of the most important changes between versions 1.x and version 2.00. To get an updated list of all changes, please read the file `CHANGES` in the distribution package.

- The *Registration procedure* to get a free *license* does no longer exists : with versions 2.x you do not have any registration form to send.
- The *sources of MegaWave2 kernel* (preprocessor and libraries) are given. Therefore, you are now able to modify the kernel, for example to adapt it on a new machine architecture.
- In return, the kernel binaries are no longer distributed : the local MegaWave2 administrator has to compile the kernel by himself.

- MegaWave2 has been adapted for *Linux* on i386.
- Effort has been done to offer more modules with associated demos, and former modules have been fully checked (some of them may see their name changed). The `src` directory has been deeply reorganized.
- The `doc` directory structure has changed : see Section 7.
- The `data` directory structure has changed, together with the search path convention. In short, a PUBLIC subdirectory has been created where all public data not specifically associated to some modules have to be put. To resolve an input filename, a module now searches inside all subdirectories of data (See Volume 2 "MegaWave2 System Library").

1.5 Contents of the MegaWave2 guides

There are four guides about MegaWave2. Here are their contents:

- **Volume 1 : "MegaWave2 User's Guide"**. This is the present guide. It introduces the user to the MegaWave2 philosophy and environment, it explains the software installation and gives the directions for use.
- **Volume 2: "MegaWave2 System Library"**. It is a reference manual on the functions of the system library. It also details all the available MegaWave2 objects. Users who do not need to write modules may skip it.
- **Volume 3: "MegaWave2 User's Modules Library"**. It is a reference manual on the public modules. This manual is automatically generated by the MegaWave2 compiler to reflect the current modules library. You can get an updated version of this volume at any time. It is essential for all users.
- **Volume 4: "XMegaWave2 User's Guide"**. (Not yet available). It is a guide for the XMegaWave extension to MegaWave2, called XMegaWave2, which is a Windowed-oriented interface and interpreter. At this time, the MegaWave2 distribution package does not include the XMegaWave extension.

1.6 What you will find in this guide

The section 1 gives you an overall idea about the software. In Section 2 you will learn how to install it and how to install the environment of each user. The section 3 guides you to write and compile your first MegaWave2 module. Next sections give references about the header's module (section 4) and the system macros (section 5). The section 7 explains how to write the documentation attached to a module. In the last section 8 you will find the instructions to register your copy of MegaWave2 and some miscellaneous references.

1.7 The MegaWave2 philosophy: basic principles

The aim of MegaWave2 is to make the coding of signal and image-oriented algorithms easier. An algorithm is implemented as a function (or a set of functions) written in C language; such a function (or set of functions) is called a **module**.

The programmer does not write a complete program: what a module becomes is the matter of the **MegaWave2 compiler**. This compiler adds input/output code to generate a run-time command: the **module's command**. The module can then be called under the shell as Unix commands (and in a compatible mode with the MegaWave1 software). But the MegaWave2 compiler can also add interface to a window-oriented interpreter (XMegaWave2); the module is then included in the interpreter as a new function. The MegaWave2 compiler does even more; see Section 1.9 page 9 for details.

What does a module's source looks like ? you can refer to Section 3 for a first example of a module. In short, the source of a module is a file (its name is the module name followed by the `.c` extension) which contains pure C instructions. It begins with a header put into comments (`/* ... */`), so the C compiler ignores it (see Section 4 for a description of the header). But the MegaWave2 preprocessor decodes this header. Informations are defined about the module, so are the author's name, the version number, etc. More important are the informations about the **usage**, that is, about the **input/output objects** (or **input/output variables**) of the module. This usage may say that a given variable (for example a `float`) is an optional input with default value 1.0, and that another variable (for example an image with gray levels stored as float numbers) is an output.

After the header comes the regular C body, where functions are defined. One function must always be present, it is the **main function** of the module and it must have the same name as the module. When we refer to input/output objects of the module, we think about the parameters of this main function. Only the main function is global (i.e. can be accessed from other modules), all other functions are local.

In the module's skeleton below, the main function is `module` and the input/output objects are `obj1,obj2,...`:

```
/*------------------ MegaWave2 Module -----------------*/
/* mwcommand
.
. [header, including usage of obj1,obj2,...]
.
*/
/*-----------------------------------------------------*/

internal_function(a1,a2,....)
.
. [definition of this function]
.

module(obj1,obj2,....)
{
  .
  .
```

```
  internal_function(b1,b2,...);
  .
  .
}
```

A module can access to the main function of any other module, following these rules:

- They are **public modules**. Such modules are available for all users; they are located in the MegaWave2 system directory (for example `$MEGAWAVE2`). It is the responsibility of the MegaWave2 administrator to manage this set of modules (see Section 2.1).
- They are **private modules**. All MegaWave2 users coding new modules generate their own private modules (located in the directory `$MY_MEGAWAVE2`). A module cannot access to another private module *from another user*. Therefore, if somebody has written a module of some interest for others, the MegaWave2 administrator should make it public.

A module belongs to a **group**. A group puts together all modules dealing about the same subject. For example, you can imagine a group `fourier` where all the algorithms about the Fourier transform are put. But you may want to distinguish between Fourier applied on one-dimensional signals and Fourier applied on images. So you can define **subgroups** of the group `fourier`, as for example `signal` and `image`: you get two groups named `fourier/signal` and `fourier/image`.

How are the groups defined ? You define a group by creating a directory corresponding to the name of the group into the existing directory `$MEGAWAVE2/src` (or `$MY_MEGAWAVE2/src`). In our former example you must create `$MEGAWAVE2/src/fourier/signal` and `$MEGAWAVE2/src/fourier/image`. The source of the modules (with `.c` extension) has to be put into those directories. With MegaWave2 versions 1.x, in addition you had to specify in the header of each module the name of the group (see Section 4): `group={"fourier/signal"}` for all modules in the directory `$MEGAWAVE2/src/fourier/signal` and `group={"fourier/image"}` for all modules in the directory `$MEGAWAVE2/src/fourier/image`. This is no longer required in MegaWave2 versions 2.x.

What is the meaning of the input/output objects ? First, they are variables of C type. All scalar types are allowed, such as `char`, `int`, `long`, `float`, ... More sophisticated types are available. They are called **MegaWave2 memory** (or **internal**) **types**, and they are always pointers to a **structure**. The structure defines the object to be processed by the module. For example, the memory type `Cimage` represents monochrome images with gray levels of (unsigned) Char values. The memory type `Curve` represents a discrete curve in the plane, etc. See the Volume two "MegaWave2 System Library" for a description of all available MegaWave2 memory types.

How do these objects live outside a module ? They are C variables until the process finishes : a module which calls another module just gives the objects as parameters of the function; into the XMegaWave2 interpreter, all commands reside in memory and therefore objects remain C variables until they are removed. When the process finishes (e.g. at the end of a module's command), the output objects have to be saved on disk as a file. The same problem occurs when a process begins: input objects have to be read from files. The format of the file depends on the memory type, we call it the **MegaWave2 file** (or **external**) **type**. Whereas there is

only one memory type associated to an object, a memory type may be represented on disk with many different file types. This is because MegaWave2 recognizes several standard file formats, especially for the images. Also see Volume two for a description of all available MegaWave2 file types.

1.8 MegaWave2 Macros

A MegaWave2 **macro** is a Bourne shell script with a normalized header. It can call a set of MegaWave2 modules (in command mode), in order to repeat a sequence of actions automatically. It can also be a **system's macro** which is an utility to manage MegaWave2.

For example, the macro `cmw2` calls the MegaWave2 Compiler, the macro `mwdoclatex` compiles the documentation of a module using LaTeX, ... Please refer to Section 5 for a list of all available system's macros. You can get an updated list of these macros by running `mwsysmaclist` or `mwdoc s` from the command line.

Notice that you can create your own macros (called **user's macros**). You may want, for example, to put in a macro a sequence of calls to some modules you frequently use. In the first versions of MegaWave2, the user's macros were put into the directory `$MY_MEGAWAVE2/shell`. Since the version 1.04 of the software, user's macros are managed in the same way as modules. There are **public user's macros** as well as **private user's macros**. The public user's macros are put into the same directories as the sources of the public modules, that is subdirectories of `$MEGAWAVE2/src`, while private user's macros are in the subdirectories of `$MY_MEGAWAVE2/src`. Please refer to Section 6 to learn more about user's macros.

1.9 The MegaWave2 Compiler

The MegaWave2 Compiler is activated by the macro `cmw2` (see description Section 5.1). Let us suppose that you are in the directory `$MY_MEGAWAVE2/src/test` (i.e. the group is `test`) and that you call `cmw2 mod` (i.e. the module is `mod`, coded in the file `mod.c`). If no error occurs, `cmw2 mod` processes the following items:

- A file object is created which contains the code of the module for the architecture of the machine; this object is added to a user library called `libmymw.a`.
- If `cmw2` has been called with the `-X` option (which means: prepare this module for XMegaWave2), an object is created which makes an interface between the module function and XMegaWave2; this object is added to a user library called `libmyxmw.a`.
- A document skeleton is created in the directory `MY_MEGAWAVE2/doc/obj`, which has the name `mod.doc`. This document will be included into the volume 3 of the MegaWave2 guides; it explains the use of the module (syntax of the command, synopsis of the function, input/output objects, ...).
- A command is generated and compiled for the architecture of the machine. The executable allows to run the module as a Unix command. The input/output variables of the module are read or written using files, keyboard or screen. The command is compiled using

standard C libraries and the following libraries: `mymw`, `sysmw`, `W_X11R4`, `mw`, `tiff` or `notiff`. They allow the module to access to functions of the System Library, of all the public modules, of all your private modules and of the Wdevice facilities (window toolkit). If there are shared libraries for your machine architecture, you will save a lot of disk space.

2 Installation

2.1 Installation of the system

This section explains how to install the MegaWave2 software on your machine, it is written for the MegaWave2 administrator. If you are a plain user, you can skip this and go to Section 2.2. To be the MegaWave2 administrator, you don't need to be root (super-user)[1]. The administrator has to compile the kernel and the modules, and to manage the local public version of the software; in addition to the installation, he may update the modules library with some private modules from the users (see Section 1.7).

2.1.1 Upgrading to a new MegaWave2 version

If a former MegaWave2 software is installed on your machine, you may upgrade to a new version by following the same instructions as for a first installation. Old public modules will be replaced by new ones. It is likely that you or other users would have developed private modules (i.e. modules set in `$MY_MEGAWAVE2/src`) that make use of older system library and older public modules. In such a case when the new installation is finished, check if public modules used by private modules have changed in some way (especially in the input/output parameters) and modify accordingly private modules. And then, recompile all private modules (run a `cmw2_all -clear -2p -dep .` under `$MY_MEGAWAVE2/src`). All MegaWave2 user's should do such a recompilation on their own private modules, even if the sources (.c files) didn't require any modification : from one MegaWave2 version to a new one, the kernel may change a lot and therefore objects (such as .o and .doc files) generated by the old kernel may not be compatible with the new one.

2.1.2 Restoration of the package

The MegaWave2 administrator has to install the distribution package in a subdirectory of its home directory. Let us call this subdirectory `$PRIVATE_MEGAWAVE2`. If needed, the administrator may also create a directory (usually outside its home directory) where the public version of the software will be copied. Let us call this subdirectory `$PUBLIC_MEGAWAVE2`.

You should begin the installation by restoring the full MegaWave2 package in `$PRIVATE_MEGAWAVE2`. As you are reading this document, you probably have already completed the restoration (in that case, you can go to the next Section 2.1.4).

If not, the best way to download the last version of the software and to get full instructions for the restoration is to visit our *World Wide WEB* page[2] on the Internet (you may also try an *anonymous ftp*[3], but this service is likely to be interrupted and the last version may not be available there). The package should be in a file named `MegaWave2_V<n>.tar.Z`, `MegaWave2_V<n>.tar.gz` or `MegaWave2_V<n>.tgz`, where `<n>` is the version number. After

[1] It is not a good idea to install MegaWave2 as root. If you have root privileges, you should better create a *megawave* account and install the software using this account.

[2] `http://www.cmla.ens-cachan.fr/Cmla/Megawave`

[3] `ftp://ftp.cmla.ens-cachan.fr`

downloading it and if your WWW browser didn't do it for you, decompress the tar file using `uncompress` (`.Z` file) or using `gunzip` (`.gz` and `.tgz` files), and extract the files from the tar file using `tar xf MegaWave2_V<n>.tar`. If you are using the GNU tar command, you may also handle restoration of `MegaWave2_V<n>.tgz` in one pass by means of the "z" flag : `tar xfz MegaWave2_V<n>.tgz`.

In any case, first load the file named README and follow the instructions in it. When the restoration will be done, you will get a main directory containing the software, with a name referred to as `$PRIVATE_MEGAWAVE2`.

Please read first the additional information put in the file `$PRIVATE_MEGAWAVE2/README`.

If you are the administrator, you should entirely read the following sections. However, you may try to install the software quickly by calling the shell `Install` located at the root directory of the MegaWave2 package `$PRIVATE_MEGAWAVE2`. If your system and machine architecture is the same as one we have, this may install the whole software without pain.

2.1.3 Directories structure

We know from Section 1.7 that modules are split between *public modules* put in a directory named `$MEGAWAVE2` and *private modules* put in `$MY_MEGAWAVE2`.

Actually, for the MegaWave2 administrator things are a little bit more complicated : the administrator may want to be able to check new public modules and new kernel binaries before making them available for plain users. Therefore, the administrator has his own `$MEGAWAVE2` directory wich is not necessary the same as the `$MEGAWAVE2` directory of plain users. His own `$MEGAWAVE2` directory is called *private MegaWave2* (with associated environment variable `$PRIVATE_MEGAWAVE2`) whereas the `$MEGAWAVE2` directory of plain users is called *public MegaWave2* (`$PUBLIC_MEGAWAVE2`). Of course, if you don't plan to update the system or if the only user is the administrator you should set `$PUBLIC_MEGAWAVE2` = `$PRIVATE_MEGAWAVE2`. Notice that the administrator always has `$PRIVATE_MEGAWAVE2` = `$MEGAWAVE2`.

2.1.4 Set up the environment

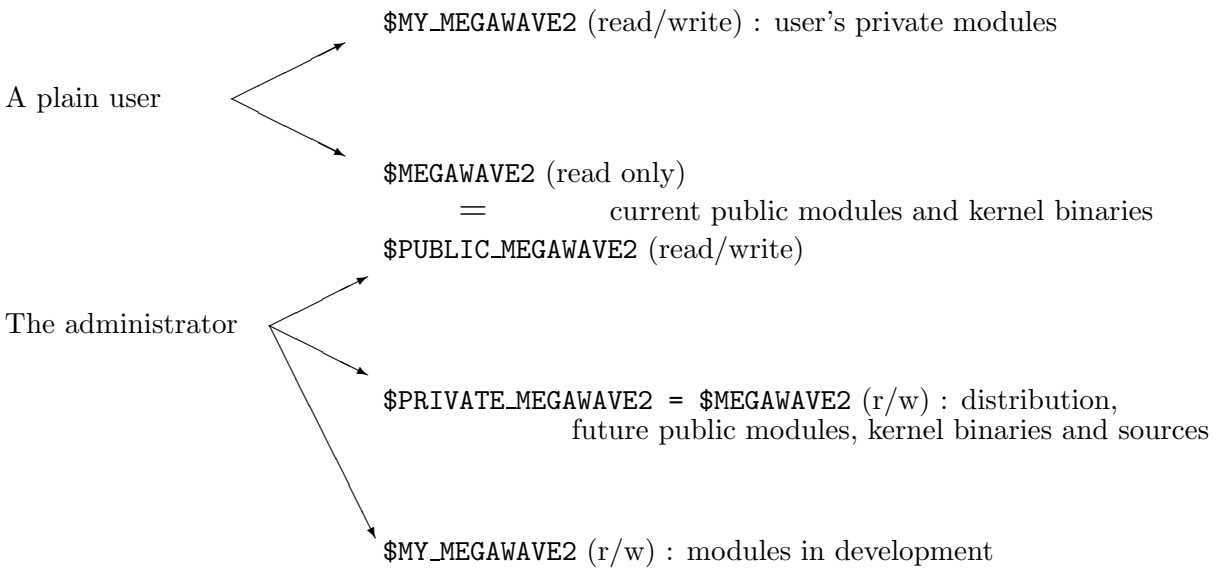
You have to set up some environment variables used by MegaWave2. Put these definitions in a file which is executed at login time (such as `.profile`,`.login`,...) or when opening a new shell (e.g. `.cshrc`). They have to be set both for the administrator account and for all user accounts. Therefore, it may be better for the administrator to put the definitions in a file which will be sourced at login time by all users.

The system's macro `mwsetenv` helps you to generate such a file, see its description in Section 5.1. This macro is called by the shell `Install` (see Sections 2.1.5 and 5.1), so if you choose to install the software in this way you should not have to set up the environment manually.

The following variables are needed *for the administrator only*:

- `PUBLIC_MEGAWAVE2` : directory where the public installation of the software has to be made, usually outside the administrator home directory. You need write permission on it.

Figure 1: The various MegaWave2 directories a user may see.



Plain users need read and execute permissions, but they should not have write permission. Example: `setenv PUBLIC_MEGAWAVE2 /usr/local/share/megawave2`.

- `PRIVATE_MEGAWAVE2` : directory where the temporary installation of the software has to be made (future public version), usually inside the administrator home directory. This is also the directory where the original distribution is put. Plain users do not need any permission on it.
  Example: `setenv PRIVATE_MEGAWAVE2 ${home}/megawave2`.

The following variables are needed for all users (including the administrator):

- `MEGAWAVE2` : for a plain user, directory where the public installation of the software is made (ask your administrator for it). For the MegaWave2 administrator, set it to `PRIVATE_MEGAWAVE2`.
  Example: `setenv MEGAWAVE2 /usr/local/share/megawave2`.

- `MY_MEGAWAVE2` : directory where the private user's version of the modules is.
  Example: `setenv MY_MEGAWAVE2 ${home}/my_megawave2`.

- `MW_MACHINETYPE` : machine architecture as returned by the macro `mwarch` (if the version of this macro accepts the option `-s`, use it).
  Example: `setenv MW_MACHINETYPE '$MEGAWAVE2/sys/shell/mwarch -s'`. On Sun computers running Solaris (SunOS 5.x or higher), you need to call `mwarch` with the option `-s`. Otherwise, MegaWave2 will confuse the objects with those for SunOS 4.x.

- `MW_SYSTEMTYPE` : name of the operating system as HPUX, SunOS, ...
  Example: `setenv MW_SYSTEMTYPE ‘uname | tr -d -‘`.

Special attention is requested if the directory path put in `MEGAWAVE2` or `MY_MEGAWAVE2` corresponds to a link or to an automounted file: you should always put the true pathname and not the link name. In short, if you type under your shell the command `cd $MEGAWAVE2` followed by the command `/bin/pwd` (and not the shell built-in command `pwd`), you must get the same pathname as the one put in `$MEGAWAVE2`.

The following variables may be needed:

- `MW_INCLUDEX11` : directory where the X Window include files are, if not in `/usr/include/X11`.
  If you are the administrator, this variable is always required.
  Example: `setenv MW_INCLUDEX11 /usr/include/X11R5`
- `MW_LIBX11` : directory where the X Window libraries are, if not in `/usr/lib/X11`. If you are the administrator, this variable is always required.
  Example: `setenv MW_LIBX11 /usr/lib/X11R5`.
- `MW_INCLUDEXm` : XMegaWave2 only. Directory where the Motif include files are, if not in `/usr/include/Xm`.
  Example: `setenv MW_INCLUDEXm /usr/include/Motif1.2`.
- `MW_LIBXm` : XMegaWave2 only. Directory where the Motif libraries are, if not in `/usr/lib`.
  Example: `setenv MW_LIBXm /usr/lib/Motif1.2`.
- `MW_LIBTIFF` : directory path where the TIFF library (libtiff) is located, if you want to use TIFF image format.
- `MW_LIBJPEG` : directory path where the JPEG library (libjpeg) is located, if you want to use JPEG image format.
- `LD_LIBRARY_PATH` : add directory path where the MegaWave2 system shared libraries are located, on UNIX systems (e.g. IRIX, SUN SOLARIS, LINUX) which require this definition for their runtime linker. Example :
  `setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${MEGAWAVE2}/sys/lib/${MW_MACHINETYPE}`.
- `LD_RUN_PATH` : add directory path where the MegaWave2 system shared libraries are located, on UNIX systems (e.g. LINUX) which require this definition for their runtime linker. Example :
  `setenv LD_RUN_PATH ${LD_RUN_PATH}:${MEGAWAVE2}/sys/lib/${MW_MACHINETYPE}`.

Note:

- The X Window system Version 11 is a priori needed only by modules which use the Wdevice library to display signals, images, movies, ... However, it cannot be discarded since it is required to compile the kernel.
- The Motif Window system is needed only for XMegaWave2; in that case you need also the X Window system Version 11.

- LaTeX is needed only to make a new documentation for the modules you are going to write. But to read the current documentation, you need a DVI viewer. MegaWave2 assumes `xdvi` (DVI Previewer for the X Window System) is installed on your system.
- The TIFF library is needed to load and save images in the TIFF format only. You may already have this library, e.g. if you use the `XV` software of John Bradley. If not, you can load it by anonymous ftp at the following Internet address: `sgi.com (directory graphics/tiff)`.
- The JPEG library is needed to load and save images in the JPEG/JFIF format, as defined by the Independent JPEG Group. Be aware that this is a loosely image format.

Last, you need to update your path variable in order to allow execution of the MegaWave2 commands from any location. Add the following paths (the first one should have greatest priority): `${MY_MEGAWAVE2}/shell`, `${MY_MEGAWAVE2}/bin/${MW_MACHINETYPE}`, `${MEGAWAVE2}/sys/bin/${MW_MACHINETYPE}`, `${MEGAWAVE2}/sys/shell`, `${MEGAWAVE2}/bin/${MW_MACHINETYPE}`.

Example:

```
set path=(${MEGAWAVE2}/sys/bin/${MW_MACHINETYPE} ${MEGAWAVE2}/sys/shell ...
          ${MEGAWAVE2}/bin/${MW_MACHINETYPE} $path)
set path=(${MY_MEGAWAVE2}/shell ${MY_MEGAWAVE2}/bin/${MW_MACHINETYPE} $path)
```

2.1.5 Make the software ready

The simplest way to install the software is to call the shell `Install` located at the root directory of the MegaWave2 package `$PRIVATE_MEGAWAVE2`, and to answer some questions. Actually, this shell simply calls the macro `mwinstall` with parameters depending on your answers (see the description of this system macro in Section 5.1). If `mwinstall` successfully exits, you should not have to do anything else to install the software but to include the environment file generated by `mwsetenv` in your `.profile` or `.cshrc`.

If your system is the same as one we have, the installation procedure should be straightforward. If not or if something goes wrong during the installation procedure, you will have to compile the whole thing manually. The following explains the main steps you should manually complete. You may also have a look on the installation shells, such as `Install` and `mwinstall`, for a better understanding of what has to be done. Once the environment variables are set (see Section 2.1.4), you should try to compile the kernel first, and the modules and user's macros afterwards.

The kernel, that is to say the system functions and the MegaWave2 preprocessor, is no longer pre-compiled on MegaWave2 versions 2.x. You may compile it for your system architecture using the shell `$PRIVATE_MEGAWAVE2/kernel/Install`.

The software contains the modules as source files (located in the directory `$MEGAWAVE2/src`), so they have to be compiled for your machine architecture. Just type `cmw2_all $MEGAWAVE2/src` to compile all the modules located in the subdirectories of `$MEGAWAVE2/src`. If your system can run the compiler, you probably won't encounter a lot of errors at this time. Most likely errors are those about standard libraries or include files not found: check the environment setup or

the installation of your operating system. If you get an error about an unsatisfied symbol while compiling a module `A`, it is probably because `A` calls a module `B` which has not been already compiled. To fix that, you may type `cmw2_all $MEGAWAVE2/src` one more time or, preferably, use the option `-2p` (two pass - does not work for all linkers, see Section 5).

2.1.6 No longer registration, but still license

When the modules are compiled, you may want to check some algorithms. Just type from your Shell the name of the module you want to run: MegaWave2 recalls you the parameters needed to execute the corresponding algorithm. Please refer to the Volume three: "MegaWave2 User's Modules Library" to learn more about the different modules. You may also run the system's macro `mwdoc M` to get the list of available modules and user's macros, together with a short description.

On MegaWave2 versions 1.x you had noticed that some input and output data were disturbed. This is because you had to register the copy of MegaWave2 you got in order to use it freely. This is no longer the case on MegaWave2 version 2.x.

However, the use of MegaWave2 is still under a license. By using it, you accept the terms of this license. You will find the license text at Section 8.1 page 79. If you do not agree with these conditions, you have to remove all the MegaWave2 files in your possession. Otherwise, your copy may be declared illegal regarding the european laws.

2.2 Installation for the user

This section explains what each user must do in order to run the software. It includes the MegaWave2 administrator which may also want to use the software as a plain user.

If you were using an older MegaWave2 software, you will have to recompile all private modules after the new software will be installed : check if some of your private modules need to be upgraded and run a `cmw2_all -clear -2p -dep .` under `$MY_MEGAWAVE2/src`. You have to redo all compilations even if your private modules didn't require any modification : from one MegaWave2 version to a new one, the kernel may change a lot and therefore objects (such as .o and .doc files) generated by the old kernel may not be compatible with the new one.

2.2.1 Set up the environment

The user may have to set up some variables of the environment. Two cases may be encountered: if the MegaWave2 administrator has put the definitions in a specific file (this is automatically done if he has used the standard installation procedure), you just have to load this file at login time or when opening a new shell (by using the shell command `source` or the `.` - dot -). Ask the local administrator about this possibility. If there is no such a file, you have to set the variables in your own configuration file as explained at Section 2.1.4 page 12.

In addition to the standard environment setup, you may want to define the following variables

- `MW_STDOUT`, `MW_STDERR`

If you type under your C-compatible shell `setenv MW_STDOUT /dev/null`, you ask the system to redirect the standard output of all modules to the null device that is, to the trash. Make the experience by calling a run-time module which usually prints a lot of messages... No more messages will disturb your terminal ! But you have lost the messages. If you want to get them in a file (let us call `output` its name), type `setenv MW_STDOUT output`. You can reset the default prints by removing your definition: `unsetenv MW_STDOUT`. You can also redirect the standard error by setting the variable `MW_STDERR`. We do not recommend you to redirect this output to the trash, since you won't be able to know if your modules correctly exit.

- `MW_CHECK_HIDDEN`
  If this variable is set, each time a module is called a check is performed using the system's macro `mwwhere` to see if the called module does not hide another one of same name. Indeed, path for private modules is set before path for public modules in the `PATH` variable (and the same rule applies for module's library at link time). So, if you write a user's module called `foo` while a `foo` public module already exists, the public module will be hidden by the first one. In such a case and providing `MW_CHECK_HIDDEN` is set, a warning message will be issued by the non-hidden module at running-time. The same procedure applies to macro, to check if a private user's macro hides a public one. However, for macros the check is performed when the header/usage text is output only (e.g. when calling the macro with -help or with an invalid syntax).

2.2.2 Make the directory tree ready

An user may want to develop new modules. In this case, he writes its private modules into a local MegaWave2 directory called `$MY_MEGAWAVE2`. This directory should contain several subdirectories although some of them are optional:

- `src` : sources of the user's modules and macros. May be divided into subdirectories representing groups.
- `bin` : executable modules. You have one subdirectory per machine architecture on which you have compiled the modules.
- `lib` : user's library of modules. You have one subdirectory per machine architecture on which you have compiled the modules.
- `obj` : object files. You have one subdirectory per machine architecture on which you have compiled the modules.
- `doc` : documentation. May be divided into `src` (source) and `obj` (object) subdirectories.
- `data` : samples of MegaWave2 data files such as images, movies, filters, shapes. May be divided into subdirectories representing groups or whatever else.
- `shell` : links to the user's macros (Bourne shell scripts).
- `mwi` : usage interface for the interpreter (XMegaWave2).

- `tmp` : directory for temporary files.

You should only focus on the directories `src`, `doc/src` and maybe `data` (the last one being optional), where you will have to write something. Other directories will be automatically updated by the system.

Subdirectories of these directories will be automatically created when necessary (except groups into `src`, see Section 1.7 page 7). But the directories themselves have to be created before the first call to the MegaWave2 compiler. You can make them manually, or you can call the macro `mwnewuser`.

3 Getting started

This section shows you how to write a MegaWave2 module, by explaining several examples. If you want to run existing modules only, please refer to the Volume three: "MegaWave2 User's Modules Library" to learn more about the different modules.

3.1 A simple module

Let us suppose that you want to write an algorithm which adds the content of two monochrome images, that is to say it adds pixel by pixel the gray level values.

The first question you should ask yourself is "what should be the input and the output of this algorithm ?". Obviously you need two images for the input and the output will be another image, which will support the result of the addition. If you refer to the Volume two: "MegaWave2 System Library", you will see that monochrome images may be implemented by two MegaWave2 objects: the memory type `Cimage` and the memory type `Fimage`. The first one uses small integers for the gray levels (unsigned char) while the second one uses floating point representation. Let us say that your algorithm will make additions using the floating point representation: this is the less restrictive choice since integers are reals (and reals are not integers !) and the addition of two integers may exceed the capacity of the unsigned char representation (255).

Other important questions are "what name should I give to the module ?", "what group should it belong to?". Due to the limitation of some standard link editors and archives, we recommend you choose short names (with no more than 11 characters). One uses to begin the name with a letter which recalls the memory type of the input: here it will be `f` since we use `Fimage`. The remainder should be chosen in order to recall the algorithm, here `add1` for example (the letter `1` means that variants will be presented). You may also want to put the module in a group, let us say `demo`.

At this point, we have all the information to write the MegaWave2 header (please have a look at the listing page 21):

- the line `/* mwcommand` tells the MegaWave2 compiler that you begin the header. It is enclosed by comments (`/* ... */`) since standard C compilers must be able to compile a module.
- the line `name = {fadd1};` defines the name of the module to be `fadd1`. Although this event is not reported as an error, this name must coincide with the name of the module file and with the name of the main function of the module.
- the line `author = {"My name"};` gives the name of the author(s). It will be used for copyrights.
- the line `labo = {"My labo with the address"}` reports the laboratory(ies) where the author(s) belong(s). It will be used for copyrights (this definition is optional. If not set, native laboratory is assumed).
- the line `version = {"1.0"};` indicates the version number of the algorithm (this definition is optional).

- the line `function = {"Adds the ... "};` explains (very) shortly the algorithm.
- Last, the more important field is the usage. Here,
  - `fimage1->A` means that `A` is an input parameter of the function `fadd1` (whose type —here `Fimage`— is given by the C declaration). The word `fimage1` is the one which will appear in the help and in the documentation (instead of the C word `A`). The string following this declaration explains shortly the meaning of this argument.
  - `fimage2->B` means that `B` is also an input parameter of the function.
  - `result<-C` means that `C` is an output parameter of the function.

To be in concordance with this header, you must edit the module in a file called `fadd1.c` into the directory `$MY_MEGAWAVE2/src/examples`. This module is actually given with the standard distribution and it is located in `$MEGAWAVE2/src/examples`. So you can avoid editing it by copying it on your home location `$MY_MEGAWAVE2/src/examples`.

After the MegaWave2 header comes the regular C body. You must first include the needed standard C files (here `stdio.h`) followed by the MegaWave2 include file (`mw.h`).

The declaration of the main function (`fadd1`) must list all the parameters (and no others) put in the `usage` part of the header: you recognize the three parameters `A,B,C`. The body of this function uses several functions of the system library (all functions of the system library begin with the prefix `mw`). Please see the Volume two: "MegaWave2 System Library" to get explanations about all these functions.

The instructions

```
if((A->nrow != B->nrow) || (A->ncol != B->ncol))
   mwerror(FATAL, 1, "The input images have not the same size!\n");
```

check that the sizes of the two input images `A` and `B` are the same, that is, that they have the same number of rows and columns. If not, a fatal error is sent, which terminates the module.

The line

```
 if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
   mwerror(FATAL, 1, "Not enough memory !\n");
```

allocates memory for the output image `C`, if `C` does not have enough memory to record a size of (`A->nrow,A->ncol`). Indeed, when you write a module, you don't know in which context the module may be executed: It can be run in the command mode, in which case no memory is allocated for `C` (but the C structure by itself) or it can be run as a function in memory (e.g. call from another module, from an interpreter, ...) in which case `C` may have been previously allocated.

The loop

```
for (x=0;x<A->ncol;x++) for (y=0;y<A->nrow;y++)
   {
```

```
      a = mw_getdot_fimage(A,x,y);
      b = mw_getdot_fimage(B,x,y);
      mw_plot_fimage(C,x,y,a+b);
    }
```

is the main part of the algorithm: for each pixel (x, y), we get in `a` the gray level of the image `A` and in `b` the gray level of the image `B` . The last instruction makes the addition `a+b` and put the result in the image `C`.

We show in the following the whole listing of the simple module called `fadd1`:

```
/*--------------------------- MegaWave2 module -----------------------------*/
/* mwcommand
 name = {fadd1};
 author = {"My name"};
 labo = {"My labo with the address"};
 version = {"1.0"};
 function = {"Adds the pixel's gray-levels of two fimages (for demo #1)"};
 usage = {
   fimage1->A
      "Input fimage #1",
   fimage2->B
      "Input fimage #2",
   result<-C
      "Output image"
};
*/
/*----------------------------------------------------------------------------*/

#include <stdio.h>
#include  "mw.h"

void fadd1(A,B,C)

Fimage  A,B,C;

{ int x,y;
  float a,b;

  if((A->nrow != B->nrow) || (A->ncol != B->ncol))
    mwerror(FATAL, 1, "The input images have not the same size!\n");

  if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  for (x=0;x<A->ncol;x++) for (y=0;y<A->nrow;y++)
    {
      a = mw_getdot_fimage(A,x,y);
      b = mw_getdot_fimage(B,x,y);
      mw_plot_fimage(C,x,y,a+b);
    }
}
```

You are now ready to compile the module: type under your favorite shell the command:
`cmw2 fadd1`
This assumes that you have a standard C compiler called `cc`. If your prefer to use the GNU C compiler, type the command:
`cmw2 -gnu fadd1`
You should get something like the following prints:

```
cmw2 fadd1

>>> Welcome on the MegaWave2 Compiler V1.18 <<<

  Phase 1 : MegaWave2 preprocessor
fadd1.c :
done.
  Phase 2 : production of sources and objects
          production of document squeleton (doc/demo/fadd1.doc)
          production of arguments analyser object
          production of module object
          add objects to MegaWave2 user library
          production of interface with MegaWave2 interpretor source
          production of interface with MegaWave2 library object
  Phase 3 : production of MegaWave command
          linking MegaWave2 command "fadd1" on hp
     (New command added... Type rehash)
```

A lot of errors may occur during this compilation if your installation is not ready. In that case, please check the installation (see section 2).

If you didn't get any errors, type `rehash` if you are using a C-shell and execute the module in the command mode by typing `fadd1` first without parameters. You should get something like:

```
------------------------------------------------------------------------------
\\     //  Adds the pixel's gray-levels of two fimages (for demo #1).
 \\   //
  fadd1    Copyright (C)1998 Jacques Froment.
 //   \\   MegaWave2 : J.Froment (C)1988-98 CEREMADE, Univ. Paris-Dauphine
//V 1.0\\  and (C)1998-2002 CMLA, ENS Cachan, 94235 Cachan cedex, France.
------------------------------------------------------------------------------

error : missing 'fimage1'

usage : fadd1 fimage1 fimage2 result

        fimage1 :       Input fimage #1
        fimage2 :       Input fimage #2
        result :        Output image
```

This print recalls you the syntax of the command. Since you run the module in the command mode, the parameters associated to a MegaWave2 type (`fimage1 fimage2 result`) refer to file

names. If you have some favorite image files available, use them for the two requested inputs `fimage1 fimage2`. If not (or if MegaWave2 cannot read those file formats), type the line:
`fadd1 fimage cimage result`
How MegaWave2 does access to the files `fimage` and `cimage` ? If a file is not found in the given path and name (here `./fimage` and `./cimage`), MegaWave2 tries to resolve the name by reading the directories `$MY_MEGAWAVE2/data` and `$MEGAWAVE2/data` and their subdirectories. For each MegaWave2 internal type, we put in those directories examples of external types. The name of the file corresponds to the name of the internal type: for example, the file
`$MEGAWAVE2/data/PUBLIC/cimage` contains a monochrome image where the gray levels were recorded using the unsigned char representation, that is the representation of the `Cimage` internal type. You can display this image on your screen by calling the following public module:
`cview cimage`

You may have noticed that we called `fadd1` with a parameter (`cimage`) which does not match the requested internal type for the C variable `B` (a `Fimage`). It doesn't matter since the MegaWave2 library makes the right conversion. It is also possible to convert a Fimage into a Cimage although is this case information may be lost. Of course some conversions are not supported (e.g. you cannot call `fadd1` with a signal as parameter). Please refer to Volume two: "MegaWave2 System Library" to learn more about that.

After the command `fadd1 fimage cimage result` has been run, you should get on disk the new file `./result` which records the contents of the C variable `C`. You can display it on the screen by typing:
`cview result`
You may get a warning message like
`MegaWave warning (cview) : 27757 Gray levels were out of [0,255].`
This is because `cview` takes a `Cimage` as the input, so the gray levels of `result` out of 255 have been thresholded.

Because `C` is a `Fimage`, the file `./result` has a compatible format with the floating point representation. By default, MegaWave2 uses the same format as the format of the input `fimage` but you can choice the format you want by using the option `-ftype`. Indeed, there are some options in addition to the options you can define in the usage. Such options are called **system's options** and one can distinguish them from **user's options** since they use more than one letter (see section 4.8 page 43 for more details). For example, `fadd1 -ftype IMG fimage cimage result2` creates a file `result2` which has the external format `IMG`. This file format uses the unsigned char representation, therefore you get this warning during the execution of `fadd1`:
`MegaWave warning (fadd1) : 27757 Gray levels were out of [0,255].`
But now if you call `cview result2`, no more warning appears since thresholding has been done by `fadd1`.

If you want to print the output image on a PostScript printer, you should use the external format `PS` instead of `IMG`: the file `result2` will contain the image following PostScript format, so you can print it using the command `lp result2` (on most Unix systems). If you don't want to print `result2` alone, e.g. if you want to include this file as an image in a LaTeX document, you should use the `EPSF` format instead of `PS`.

3.2 Some optimizations

The main part of the algorithm in `fadd1` uses standard MegaWave2 functions to access to the pixels. Since fast pixel access is critical for image processing, we give in this section two alternatives to speed the execution of the module. You may find more tips in the Volume two: "MegaWave2 System Library", at this time keep just in mind that this kind of optimization can be adapted to other memory types such signals or movies.

The first alternative is given by the module `fadd2`. Here is the listing:

```
/*-------------------------- MegaWave2 module -----------------------------*/
/* mwcommand
 name = {fadd2};
 author = {"Jacques Froment"};
 version = {"1.0"};
 function = {"Adds the pixel's gray-levels of two fimages (for demo #2)"};
 usage = {
   fimage1->A
      "Input fimage #1",
   fimage2->B
      "Input fimage #2",
   result<-C
      "Output image"
};
*/
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include  "mw.h"

#define _(a,i,j) ((a)->gray[(j)*(a)->ncol+(i)])

void fadd2(A,B,C)

Fimage  A,B,C;

{ int x,y;

  if((A->nrow != B->nrow) || (A->ncol != B->ncol))
    mwerror(FATAL, 1, "The input images have not the same size!\n");

  if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  for (x=0;x<A->ncol;x++) for (y=0;y<A->nrow;y++)
    _(C,x,y) = _(A,x,y) + _(B,x,y);
}
```

This module does not really optimize the speed of the execution (the saving is small) but it makes the listing more readable for people: once you have defined the C macro `_(a,i,j)`, you can access to the pixel (x, y) on an image `I` by writing `_(I,x,y)` only. But be aware that some

checking are deconnected in this way: do not exceed the maximum range for (x, y).

The second alternative is given by the module `fadd3`. This one really improves the speed of the execution. But and unlike the former example, it cannot be used in all algorithms: the pixel cannot be addressed in random order. The image has to be read (or written) in the "natural" order that is, from up to down and left to right. Here is the listing:

```
/*--------------------------- MegaWave2 module -----------------------------*/
/* mwcommand
 name = {fadd3};
 author = {"Jacques Froment"};
 version = {"1.0"};
 function = {"Adds the pixel's gray-levels of two fimages (demo #3)"};
 usage = {
   fimage1->A
      "Input fimage #1",
   fimage2->B
      "Input fimage #2",
   result<-C
      "Output image"
};
*/
/*--------------------------------------------------------------------------*/

#include <stdio.h>
#include  "mw.h"

void fadd3(A,B,C)

Fimage  A,B,C;

{
  register float *ptr1,*ptr2,*ptr3;
  register int i;

  if((A->nrow != B->nrow) || (A->ncol != B->ncol))
    mwerror(FATAL, 1, "The input images have not the same size!\n");

  if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  for (ptr1=A->gray, ptr2=B->gray, ptr3=C->gray, i=0;
       i < A->nrow*A->ncol; ptr1++, ptr2++, ptr3++, i++)
    *ptr3 = *ptr1 + *ptr2;
}
```

The third alternative is given by the module `fadd4`. The speed of the execution for this module is not so fast that the one for the previous module `fadd3`, but this last method can be used in any algorithms, even if the pixels are not scanned in the "natural" order. This method consists to allocate, using a MegaWave2 function, a bi-dimensional tab which is actually an one-dimensional tab of pointers, and each pointer points to the beginning of a image's line. Here is the listing:

```
/*--------------------------- MegaWave2 module -----------------------------*/
/* mwcommand
 name = {fadd4};
 author = {"Jacques Froment"};
 version = {"1.0"};
 function = {"Adds the pixel's gray-levels of two fimages (for demo #4)"};
 usage = {
   fimage1->A
      "Input fimage #1",
   fimage2->B
      "Input fimage #2",
   result<-C
      "Output image"
};
*/
/*--------------------------------------------------------------------------*/

#include <stdio.h>
#include  "mw.h"

void fadd4(A,B,C)

Fimage  A,B,C;

{ int x,y;
  float **TA,**TB,**TC;

  if((A->nrow != B->nrow) || (A->ncol != B->ncol))
    mwerror(FATAL, 1, "The input images have not the same size!\n");

  if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  if ( ((TA = mw_newtab_gray_fimage(A)) == NULL) ||
       ((TB = mw_newtab_gray_fimage(B)) == NULL) ||
       ((TC = mw_newtab_gray_fimage(C)) == NULL) )
    mwerror(FATAL, 1, "Not enough memory !\n");

  for (x=0;x<A->ncol;x++) for (y=0;y<A->nrow;y++)
    TC[y][x] = TA[y][x] + TB[y][x];

  free(TC);
  free(TB);
  free(TA);
}
```

3.3 A little more complex extension

We give in this section a little more complex extension to our algorithm, in order to show the easy way in which two modules can communicate. This will also gives examples of the use of

options in the usage.

You have seen in section 3.1 that the addition of two floating point images may be source of errors when the resulting image has to be converted to the unsigned char representation (which may be necessary, for example, when the image has to be displayed). You may want to control the conversion into the addition module. Let us suppose that you want to put an optional thresholding of the output image, which may be preceded by a normalization of the gray levels into a given interval [min, max].

You have to add in the new module inputs for the boundaries of the interval, and input for a flag to say whether or not you want the normalization. You have therefore to familiarize yourself with the options.

There is a general rule about the input/output optional parameters of the module function: they have to be pointers. If the pointer is `NULL` it means that the parameter has not been selected, if the address is not equal to `NULL` it points to an allocated space which may be a scalar or a C structure (in case of MegaWave2 memory type). Because all MegaWave2 memory types (e.g. `Cimage`, `Fimage`, ...) are pointers to a structure, there is no difference in the type between an optional parameter and a needed parameter. On the contrary, if the input/output optional parameter is a scalar, say a `float`, it has to be defined has a pointer to a float (`float *`).

A flag is nothing more than a pointer to any scalar, e.g. a `char *`. In the following version of our module, called `fadd`, the flag `norm` indicates if the normalization has to be computed (see the listing page 28) :

```
'n'->norm "Normalize pixel values into [min,max]"+
```

The letter `'n'` means that, in the command mode, the flag is set when the module is called with the user option `-n`.

The other options are about the boundaries, let have a look to the first one:

```
'm':min->m0 "Force minimal pixel value",
```

There is here an additional term, `min`, which says that the option needs a value when it is selected. This value is put into the C variable addressed by `m0`. The word `min` is the one which will appear in the help and in the documentation, instead of the name of the C variable (you can choice the same name). The string following this declaration explains the meaning of the option. Everything is similar for the second boundary option:

```
'M':max->m1  "Force maximal pixel value",
```

You have of course to update the declaration of the module function, in order to add the options:

```
void fadd(A,B,C,norm,m0,m1)

Fimage  A,B,C;
char *norm;
float *m0,*m1;
```

When the `-n` option is not selected, it should be possible to give only one boundary of the interval [min, max]. Indeed, the threshold may be done only in one side. But when the user ask for the normalization, both `-m` and `-M` options must be selected. So you need to check that somewhere. It is not possible to put this information in the header (we could do that but the header's grammar would be horrible), but you can add the verification in the beginning of the function:

```
if (norm && (!m0 || !m1)) mwerror(USAGE,0,
          "Normalization needs selection of [min,max] values\n");
```

We call the function `mwerror` with the argument `USAGE` to tell MegaWave2 that this error is about the usage.

The normalization and thresholding process is called by the last line:

```
if (m0 || m1) fthre(C,C,norm,m0,m1);
```

Where is the function `fthre` defined ? Since it is not a function of the system library (it doesn't begin with `mw`), and because this function is not defined in the module, it should be the main function of another module. We could have put the computations into `fadd`, but by creating another module we offer the possibility to directly use the command `fthre` in other contexts. In addition, other modules may want to make normalization or thresholding so the code will not be duplicated. Such a (well-known) "philosophy of overlapped black boxes" saves space and time, it is one of the important aspect of MegaWave2: if your mathematical algorithm may be decomposed into several independant algorithms, write one module per algorithm.

Now we rapidly study the module `fthre`. You should refer to the listing page 29. The header of `fthre` is copied from the one of `fadd`, but there is only one input image instead of two. Please notice that the output image (the C variable `B`) is a priori not the same as the input image `A`. This allows to call `fthre` from a module which needs to keep the original image. But you can force `fthre` to use the same variable for the input and the output, by calling it with twice the same variable: that is what we do in `fadd`. In this way, you save memory since the instruction

```
if ((B = mw_change_fimage(B,A->nrow,A->ncol)) == NULL)
  mwerror(FATAL, 1, "Not enough memory !\n");
```

won't allocate any memory (`A` = `B` and therefore `B` has the gray plane already allocated for the requested size).

Here is the listing of the addition module `fadd`. It is a public module put into the group `common/float_image`:

```
/*--------------------------- MegaWave2 module  -----------------------------*/
/* mwcommand
 name = {fadd};
 author = {"Jacques Froment"};
 version = {"1.0"};
 function = {"Adds the pixel's gray-levels of two fimages"};
 usage = {
```

```
 'n'->norm "Normalize pixel values into [min,max]",
 'm':min->m0 "Force minimal pixel value",
 'M':max->m1  "Force maximal pixel value",
 fimage1->A
   "Input fimage #1",
 fimage2->B
   "Input fimage #2",
 result<-C
      "Output image"
};
*/
/*-------------------------------------------------------------------------*/

#include <stdio.h>
#include  "mw.h"

void fadd(A,B,C,norm,m0,m1)

Fimage  A,B,C;
char *norm;
float *m0,*m1;

{
  register float *ptr1,*ptr2,*ptr3;
  register int i;

  if (norm && (!m0 || !m1)) mwerror(USAGE,0,
            "Normalization needs selection of [min,max] values\n");

  if((A->nrow != B->nrow) || (A->ncol != B->ncol))
    mwerror(FATAL, 1, "The input images have not the same size!\n");

  if ((C = mw_change_fimage(C,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  for (ptr1=A->gray, ptr2=B->gray, ptr3=C->gray, i=0;
       i < A->nrow*A->ncol; ptr1++, ptr2++, ptr3++, i++)
    *ptr3 = *ptr1 + *ptr2;

  if (m0 || m1) fthre(C,C,norm,m0,m1);
}
```

Here is the listing of the threshold module `fthre`. It is a public module put into the group `common/float_image`:

```
/*--------------------------- MegaWave2 module -----------------------------*/
/* mwcommand
 name = {fthre};
 author = {"Jacques Froment"};
 version = {"1.0"};
 function = {"Threshold the pixel's gray-levels of a fimage"};
```

```
 usage = {
 'n'->norm    "Normalize pixel values into [min,max]",
 'm':min->m0  "Force minimal pixel value",
 'M':max->m1  "Force maximal pixel value",
 fimage->A    "Input fimage",
 result<-B    "Output image"
};
*/
/*-----------------------------------------------------------------------*/

#include <stdio.h>
#include <math.h>
#include  "mw.h"

void fthre(A,B,norm,m0,m1)

Fimage  A,B;
char *norm;
float *m0,*m1;

{
  register float *ptr;
  register int i;
  float min,max,a,b;

  if (!m0 && !m1) mwerror(USAGE,0,"At least min or max pixel value requested\n");
  if (norm && (!m0 || !m1)) mwerror(USAGE,0,"Normalization needs selection of [min,max] values\n");
  if (m0 && m1 && (*m1 <= *m0)) mwerror(USAGE,0,"Illegal values of [min,max]\n");

  if ((B = mw_change_fimage(B,A->nrow,A->ncol)) == NULL)
    mwerror(FATAL, 1, "Not enough memory !\n");

  mw_copy_fimage(A,B);  /* Copy pixel values of A into B */

  if (norm)  /* Normalization */
    {
      min=1e20; max=-min;
      for (ptr=B->gray, i=0;  i < B->nrow*B->ncol; ptr++, i++)
        {
          if (*ptr < min) min=*ptr;
          if (*ptr > max) max=*ptr;
        }
      if (fabs((double) max-min) <= 1e-20)
        mwerror(FATAL,1,"Cannot normalize: constant input image\n");
      a = (*m1-*m0)/(max-min);
      b = *m0 - a * min;
      for (ptr=B->gray, i=0;  i < B->nrow*B->ncol; ptr++, i++)
        *ptr = a * *ptr + b;
    }

  /* Thresholding */
  if (m0) for (ptr=B->gray, i=0;  i < B->nrow*B->ncol; ptr++, i++)
```

```
    if (*ptr < *m0) *ptr=*m0;
  if (m1) for (ptr=B->gray, i=0;  i < B->nrow*B->ncol; ptr++, i++)
    if (*ptr > *m1) *ptr=*m1;
}
```

In order to make a run-time executable for the module `fadd`, you have to take the following precaution: when you want to compile a module `X` (here `fadd`) which call the function of another module `Y` (here `fthre`), be sure to compile `Y` before `X`. If you don't, you may get this kind of message during the compilation of `X`:

```
  Phase 3 : production of MegaWave command
          linking MegaWave2 command "X" on hp
/bin/ld: Unsatisfied symbols:
   Y (code)
          Error : exit.
```

There is an eviler possibility: if you change the content of the module `Y`, think to recompile `X` after `Y` (no message will be displayed if you forget that). Therefore, a good habit is to recompile all your modules from time to time (use the macro `cmw2_all` for that). You may also use the Unix *make* utility.

4 Module's header

4.1 Syntax

Any module begins with a MegaWave2 header. Such header is a set of statements enclosed between `/* mwcommand` and `*/`.

There are some restrictions :

- no C code nor C preprocessor instructions (`#include`, `#define`, ...) must be present before the MegaWave2 header,
- no C comments inside the MegaWave2 header.

The following sections describes the statements which are used in the MegaWave2 header.

Needed statements are :

- `name`,
- `author`,
- `function`,
- `usage`,
- `version`.

Optional statements are :

- `labo`,
- `group`.

4.1.1 Name Statement

Syntax : `name = {`*C_name*`};`

Description : gives the name of the C module and the name of the run-time command; *C_name* must follow the C identifier syntax. The file name of the module must be the same, but it is followed by the extension `.c`.

Example : `name = {bigtest};`

4.1.2 Author Statement

Syntax : `author = { "`*author name*`", " `*author name*`", ... };`

Description : sets the list of author names; each author name must be enclosed between quotes and be separated by a comma. These list is used to insert copyright in the run-time command and in the module documentation.

Example : `author = {"A.Turing"};`

4.1.3 Function Statement

Syntax : `function = { "`*sentence*`"};`

Description : Explains shortly the function of the module; this statement contains one string enclosed between quote. It is used every time a short help is requested.

Example : `function = { "Return 1 if a MegaWave2 module is useful, 0 elsewhere" };`

4.1.4 Labo Statement

Syntax : `labo = { "`*Laboratory name and address* `"};`

Description : sets the name of the laboratory (or firm) from which the authors are issued, followed by the address. This statement contains one string enclosed between quote. Several laboratories may be put in the same string. This string is used to insert copyright in the run-time command and in the module documentation.

Example : `labo = { "Babbage Inc. 2001 W 99th St. NY New York 10021, USA"};`

This statement is an optional statement. If not set, a default laboratory is used for the copyright (the one from which MegaWave2 is issued).

4.1.5 Group Statement

Syntax : `group = { "` *group name* `"};`

Description : sets the group name in which the module belongs; a group is a word that follows UNIX file naming syntax (including an optional path).

Example : `group = {"autoref/tests"};` Here, `tests` is a subgroup included in the main group `autoref`.

This statement is an optional statement. If not set, the default group is given by the location of the module file into the directory `$MY_MEGAWAVE2/src`. If set, it must match this location. Since MegaWave2 versions 2.x, use of this statement is no more recommended. It is still available for backward compatibility only.

4.1.6 Version Statement

Syntax : `version = { "` *version* `"};`

Description : sets the version of the module. Each time you modify the module file you should increment this number.

Example : `version = { "1.00"};`

Since MegaWave2 versions 2.x, this statement is no more optional.

4.1.7 Usage Statement

Syntax : `usage = {` *usage specification list* `};`

Description : describes the module interface for a call by the run-time command. An usage specification is made by an argument declaration followed by space and a quoted string; the quoted string is a description of the corresponding argument. The usage specifications are separated by a comma. Therefore, an extended description of the syntax may be:

```
usage = { arg_1 "string_1",
          arg_2 "string_2",
              ...
          arg_n "string_n"
        };
```

There are five types of argument declaration:

1. **options** : they are optional parameters, which are selected by putting in the command line a `-` followed by any printable character (but `?`). These character defines the option. Unix uses such options for its commands, as the option `-a` (list all entries) of the command `ls`. Options are always placed before needed arguments.

2. **needed arguments** : they are needed parameters, as the Unix command `write` has a needed arguments (which is a user name). Any module must define at least one needed argument.

3. **optional arguments** : it is a list of parameters which is used entirely or is not used at all (you cannot use some of the arguments of this list and don't use the others); this list is put between `{` and `}` and is always placed after needed arguments.

4. **variable argument** : this argument corresponds to a list of undefined length, the module loops on all arguments of the list; it is always placed after needed arguments; You cannot define for the same module optional and variable arguments.

5. **unused arguments** : this list describes the parameters of the module function which are not used as arguments in the command mode. This list should be put at the end of the declaration.

The corresponding C variables (which are defined in the module function) can be of scalar type (`short`, `int`, `long`, `unsigned short`, `unsigned int`, `unsigned long`, `float`, `double`) or of MegaWave2 type (a pointer to a structure, see the Volume two: "MegaWave2 System Library" for a list of all available MegaWave2 types).

The common description for the usage specification is the following: *H name* `->` or `<-` *C name*. The right arrow `->` describes an input flow and the left arrow `<-` describes an output flow.

The word *H name* contains the name of the argument which is used by Human being (e.g. used in the help, in the documentation) The word *C name* contains the name of the C variable used in the module body. One can choice to give to *H name* and *C name* the same word.

The syntax for each type of argument declaration is described in the next sections 4.2 to 4.6. We write by `H_id` any word of type *H name* and by `C_id` any word of type *C name*.

4.2 Options

'c'$->$`C_id`

(where 'c' is a character) defines a flag option (i.e. an option without value); `C_id` is a C variable of type pointer to a scalar (e.g. char *). The flag has been set by the user if and only if `C_id` $\neq$ `NULL`.

'c':`H_id` $->$`C_id`

(where 'c' is a character) defines an option with input value. The flag has been set by the user if and only if `C_id` $\neq$ `NULL`. If `C_id` $\neq$ `NULL`, the value of the option is given by `*C_id`. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`), in this case its value is defined in the command line by writting the number (or character) on the terminal, or a C variable of MegaWave2 type (e.g. `Cmovie`), in this case its value is given by writting the file name on the terminal.

'c':[`H_id` = `Val`]$->$`C_id`

defines an option with default input value. This case is similar to the former, but when the option is not set by the user then `*C_id`= `Val` that is, one can never have `C_id` = `NULL`. MegaWave2 types are not allowed. `Val` must be a constant of same type than `C_id`. Warning: no implicit conversions are made (e.g. if `C_id` is of type `float`, `Val` = 1.0 is allowed but not `Val` = 1).

'c':`H_id` $->$`C_id` [`Min`,`Max`]

defines an option with input value and with interval checking (an error is send if the input value does not fit into the given interval [`Min`,`Max`]). MegaWave2 types are not allowed. `Min` and `Max` must be constant of same type than `C_id` with `Max` > `Min`.

'c':[`H_id` = `Val`] $->$`C_id` [`Min`,`Max`]

defines an option with default input value and with interval checking.

'c':`H_id` $<-$`C_id`

(where 'c' is a character) defines an option with output value. The flag has been set by the user if and only if `C_id` $\neq$ `NULL`. If `C_id` $\neq$ `NULL`, MegaWave2 writes the output value at the end of the module from the content of `*C_id`. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`) or a C variable of MegaWave2 type (e.g. `Cmovie`). In this last case and if `C_id` $\neq$ `NULL`, you must allocate the structure to the right size using functions of the system library, before doing any computation with this variable (see Volume two: "MegaWave2 System Library").

Restriction: `C_id` cannot be a function (return value of the module), whereas this is allowed with needed arguments (see section 4.3).

4.3 Needed arguments

`H_id ->C_id`

defines a needed argument with input value. `C_id` is a C variable of scalar type (e.g. `char`), in this case its value is defined in the command line by writting the number (or character) on the terminal, or a C variable of MegaWave2 type (e.g. `Cmovie`), in this case its value is given by writting the file name on the terminal. In the body of the module function, you are sure to be able to access to the content of this variable. Do not perform any deallocation on it, since MegaWave2 may try to access it after the end of the module.

`H_id ->C_id[Min,Max]`

defines a needed argument with input value and with interval checking (an error is send if the input value does not fit into the given interval [`Min`,`Max`]). MegaWave2 types are not allowed. `Min` and `Max` must be constant of same type than `C_id` with `Max` > `Min`.

`H_id <-C_id`

defines a needed argument with output value. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`) or a C variable of MegaWave2 type (e.g. `Cmovie`). It can be also the return of the module (e.g. `H_id <-fentropy` if `fentropy` is the module function which returns a value of the same type than `C_id`), although this form is not always recommanded when the return value is a MegaWave2 object (since the structure has to be allocated at each call to the module). The pointer `C_id` can never be `NULL` (MegaWave2 allocates space for the value, if needed). If `C_id` is of MegaWave2 type, you must allocate the structure to the right size using functions of the system library, before doing any computation with this variable (see Volume two: "MegaWave2 System Library"). MegaWave2 writes the output value at the end of the module from the content of `*C_id`.

Caution: when `C_id` is a pointer to a scalar, there is no number (nor character) to write in the command line, and therefore these needed argument is virtual (it does not appear in the usage print).

4.4 Optional arguments

The definition of optional arguments and needed arguments differs only in the fact that the first are enclosed into `{` and `}`. In addition, optional arguments must be defined after needed arguments. When they are several optional arguments, they are linked in the sense that the selection of the first leads the need of the selection of the others.

`H_id ->C_id`

defines an optional argument with input value. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`), or a C variable of MegaWave2 type (e.g. `Cmovie`). If `C_id` = `NULL`, then the argument has not been selected.

`H_id ->C_id [Min,Max]`

defines an optional argument with input value and with interval checking (an error is send if the input value does not fit into the given interval [`Min`,`Max`]). MegaWave2 types are not allowed. `Min` and `Max` must be constant of same type than `C_id` with `Max` > `Min`.

`[H_id = Val] ->C_id`

defines an optional argument with default input value. When the argument is not set by the user then `*C_id= Val` that is, one can never have `C_id` = `NULL`. MegaWave2 types are not allowed. `Val` must be a constant of same type than `C_id`. Warning: no implicit conversions are made (e.g. if `C_id` is of type `float`, `Val` = 1.0 is allowed but not `Val` = 1).

`[H_id = Val ] ->C_id [Min,Max]`

defines an optional argument with input value and with interval checking (an error is send if the input value does not fit into the given interval [`Min`,`Max`]). MegaWave2 types are not allowed. `Min` and `Max` must be constant of same type than `C_id` with `Max` > `Min`.

`H_id <-C_id`

defines an optional argument with output value. The argument has been given by the user if and only if `C_id` ≠ `NULL`. If `C_id` ≠ `NULL`, MegaWave2 writes the output value at the end of the module from the content of `*C_id`. `C_id` is a C variable of MegaWave2 type only (e.g. `Cmovie`). If `C_id` ≠ `NULL`, you must allocate the structure to the right size using functions of the system library, before doing any computation with this variable (see Volume two: "MegaWave2 System Library").

Restriction: `C_id` cannot be a function (return value of the module), whereas this is allowed with needed arguments (see section 4.3).

4.5 Variable arguments

`...->C_id`

defines variable arguments with input values. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`), in this case its successive values are defined in the command line by writting the numbers (or characters) on the terminal, or a C variable of MegaWave2 type (e.g. `Cmovie`), in this case its successive values are given by writting the file names on the terminal. The module function is call as many time as the number of variable parameters given in the command line. If no variable parameters are given, the module function is called only once with `C_id` = `NULL`.

`...->C_id [Min,Max]`

defines variable arguments with input values and with interval checking (an error is send if one input value does not fit into the given interval [`Min`,`Max`]). MegaWave2 types are not allowed. `Min` and `Max` must be constant of same type than `C_id` with `Max` > `Min`.

`...<-C_id`

defines variable arguments with output values. `C_id` is a C variable of MegaWave2 type only (e.g. `Cmovie`): its successive values will be put in the files whose names are written on the terminal. The module function is call as many time as the number of variable parameters given in the command line. If no variable parameters are given, the module function is called only once with `C_id = NULL`.

Restriction: `C_id` cannot be a function (return value of the module), whereas this is allowed with needed arguments (see section 4.3).

4.6 Unused arguments

`notused ->C_id`

defines an argument which is not used in the command line mode. We have always `C_id = NULL` when the module function is called in the command line mode. Therefore, a value $\neq$ `NULL` means a call from the library. `C_id` is a C variable of type pointer to a scalar (e.g. `char *`), or a C variable of MegaWave2 type (e.g. `Cmovie`).

Remarks:

- The notation `->` does not mean here that the argument is an input argument.
- The word `notused` being a keyword, you cannot use it elsewhere.
- Unused arguments should be put at the end of the usage declaration.

4.7 Examples of headers

We give in the following the source of several modules, which were written in order to show the various possibilities of the header's syntax. All of these modules are available with the public distribution, in the directory `$MEGAWAVE2/src/demo`.

4.7.1 Use of options

The module `demohead1` demonstrates all the different kinds of options. It uses also two needed arguments, one in input (a scalar with interval checking) and one in output (a MegaWave2 type).

Notice that the output variable (a `Cimage`) is not a parameter of the module function: it is notified to the mother procedure using the return value of the module. This form is not recommended since it leads to allocate memory for the `Cimage` at each call to the module. The recommended form is to put the variable as a regular parameter of the module function, so the variable is allocated by a call to `mw_change_cimage(output,nrow,ncol)` only if no previously memory allocation was done.

```
/*----------------------------- MegaWave Module -----------------------------*/
```

```
/* mwcommand
  name = {demohead1};
  version = {"1.0"};
  author = {"Jacques Froment"};
  function = {"Demo of MegaWave2 header - #1 : options -"};
  usage = {
  'a'->flg
     "Flag -option-",
  'B':cimage_input_opt -> B
     "Input MegaWave2 type (cimage) -option-",
  'c':[c_opt=1.0] -> c
     "Input scalar (float) with default value -option-",
  'd':d_opt -> d          [-10,10]
     "Input scalar (integer) with boundary -option-",
  'e':[e_opt=0.0] -> e    [0.0,1e20]
     "Input scalar (double) with default value and boundary -option-",
  'F':f_opt <- F
     "Output MegaWave2 type (cimage) -option-",

  float_input -> input    [-1.0,1.0]
     "Input scalar (float) with boundary -needed argument-",

  cimage_output<-demohead1
     "Output MegaWave2 type (cimage) -needed argument-"
          };
*/
/*------------------------------------------------------------------------*/

#include <stdio.h>

/* Include always the MegaWave2 Library */
#include "mw.h"

Cimage demohead1(flg,B,c,d,e,F,input)

char *flg;   /* Or int *flg, ... */
Cimage B;    /* You don't need *B since Cimage is of MegaWave2 type (pointer)*/
float *c;    /* You need *c since float is a scalar (not a pointer) */
int *d;      /* You need *d since int is a scalar (not a pointer) */
double *e;   /* You need *e since double is a scalar (not a pointer) */
Cimage F;    /* You don't need *F since Cimage is of MegaWave2 type */
float input; /* You don't need any pointer since it's a needed input */


{
  Cimage output;  /* return of the module */

  if (flg) printf("flg flag active\n"); else printf("flg flag not active\n");

  /* B may be NULL */
  if (B)
    {
```

```
      printf("Optional input B image selected\n");
      /* Here you can access to the content of the image B
         ...
       */
    }
  else
    {
      printf("No B image selected\n");
      /* Do not access to B */
    }


  /* c cannot be NULL since *c has a default value */
  printf("*c = %f\n",*c);


  /* d may be NULL */
  if (d) printf("*d = %d\n",*d); else printf("No d value\n");

  /* e cannot be NULL since *e has a default value */
  printf("*e = %lf\n",*e);

  /* F may be NULL */
  if (F)
    {
      printf("Optional output F image selected\n");
      /* Here you can compute the image F, after dimensioning.
      */
      F = mw_change_cimage(F,10,10); /* for a size of (10,10) */
      if (F == NULL) mwerror(FATAL,1,"Not enough memory\n");
      /*
         ...
      */
    }
  else printf("No optional output F selected\n");

  /* Needed scalar argument is not a pointer */
  printf("input = %f\n",input);

  /* We need to create the structure and to allocate the output image */
  /* - In this example the size is (1,1) */
  output = mw_change_cimage(NULL,1,1);
  if (output == NULL) mwerror(FATAL,1,"Not enough memory\n");

  /* Here you can compute the image output
     ...
  */

  return(output);
}
```

4.7.2 Use of optional arguments

With the module `demohead2`, we show the use of optional arguments. This module defines also a needed output argument (a scalar).

```
/*----------------------------- MegaWave Module -----------------------------*/
/* mwcommand
  name = {demohead2};
  version = {"1.0"};
  author = {"Jacques Froment"};
  function = {"Demo of MegaWave2 header - #2 : optional arguments -"};
  usage = {

  output0 <- out0
     "Output scalar (int) -needed argument-",

     {
     [input0=1]  -> in0 [-5,5]
       "Input scalar (int) with default value and boundary - optional argument -",
     input1 -> in1
       "Input scalar (float) - optional argument -",
     output1 <- out1
       "Output MegaWave2 type (cimage) - optional argument -"
     }
           };
*/
/*---------------------------------------------------------------------------*/

#include <stdio.h>

/* Include always the MegaWave2 Library */
#include "mw.h"

void demohead2(out0,in0,in1,out1)

int *out0;    /* You need a pointer since it's an output */
int *in0;     /* You need a pointer since it's an optional argument */
float *in1;   /* You need a pointer since it's an optional argument */
Cimage out1;  /* out1 is already a pointer since it's of MegaWave2 type */

{
  /* out0 cannot be NULL */
  *out0 = 3;

  /* in0 cannot be NULL */
  printf("*in0 = %d\n",*in0);

  /* in1 may be NULL */
  if (in1) printf("*in1 = %f\n",*in1); else printf("No in1 value\n");

  /* out1 may be NULL */
```

```
  if (out1)
    {
      /* Here you can compute out1, after dimensionning */
      out1 = mw_change_cimage(out1,10,10); /* for a size of (10,10) */
      if (out1 == NULL) mwerror(FATAL,1,"Not enough memory\n");
      /*
         ...
      */
    }
  else printf("No optional output image out1\n");

}
```

4.7.3 Use of variable and unused arguments

The module `demohead3` shows the use of variable arguments. It includes also one unused argument, `Win`, which is a pointer to a scalar. The type of this argument should be `Wframe`, a pointer to a window structure (See the Volume two: "MegaWave2 System Library" about the Wdevice library). But `Wframe` is not a MegaWave2 object, therefore this type cannot be used in the header. By casting the variable in the call to `demohead3`, one can define `Win` as a pointer to any scalar.

Notice that the variable `Output` is of type `Cimage *`; it is therefore a pointer of pointer. Why ? this is not requested by the MegaWave2 header, but by the last instruction
`*Output = Input;`
This instruction changes the address pointed out by `*Output`. If `Output` was of type `Cimage`, the instruction
`Output = Input;`
would change the address inside the module function only, not outside. With `Output` of type `Cimage`, the only solution is to copy the content of `Input` into `Output` (using the function `mw_copy_cimage()`).

```
/*------------------------------ MegaWave Module -------------------------------*/
/* mwcommand
  name = {demohead3};
  version = {"1.0"};
  author = {"Jacques Froment"};
  function = {"Demo of MegaWave2 header - #3: variable and notused arguments -"};
  usage = {
    A->Input            "Input (could be a cimage)",
    ...<-Output         "Output (copy of the input)",
    notused -> Win      "Window (internal use only)"
          };
*/
/*------------------------------------------------------------------------------*/

#include <stdio.h>

/* Include always the MegaWave2 Library */
```

```
#include "mw.h"

/* Include the window since we use windows facility */
#include "window.h"

void demohead3(Input,Output,Win)

Cimage Input;
Cimage *Output;  /* Here we define *Output since the function changes the  */
                 /* pointer value (we set bellow *Output = Input)          */

char *Win;       /* Should be "Wframe *Win" for a MegaWave2 window         */
                 /* BUG: We cannot use other type than scalar or MegaWave2 */
                 /* Don't forget to cast before the function call          */
{
  if (Win != NULL)
    {
      printf("Library call: passing Window ptr\n");
      /*
         ...
      */
    }
  else printf("Command call: no Window ptr\n");

  if (*Output == NULL) printf("No output requested !\n");
  else
    *Output = Input;
}
```

4.8 System's options

The options defined in the header's module (see section 4.2) are called *user's options* since they are defined by the user. You may notice that one letter only is used to select a user's option in the command line.

In addition to user's options, there exist other options called system's options since they are defined by MegaWave2 and always available. System's options use more than one letter, therefore no confusion can be made.

Here is the list of all available system's options:

`-debug` actives the debugging mode that is, the print of the function `mwdebug` (see the Volume two for a description of this function).

`-fsum` prints the module's function summary as given in the Volume three: "MegaWave2 User's Modules Library", and exits. This recalls the user the parameters needed to call the module from another module.

`-ftype "file type"` allows to overwrite the default file type for the MegaWave2 outputs. By default, MegaWave2 selects the same file type than the one of the inputs (if the internal input type matches the internal output type) or the file type which it considers to be the "most

common". Using this option, you force MegaWave2 to write files having the type given in the string (file types are given in the Volume two: "MegaWave2 System Library"). Do not use this option in case of several outputs with incompatible types. On some file types, option(s) may be added. Option mark is ':'. For example, JFIFC:80 means JPEG/JFIFC file format with the option 80 (quality factor).

`-help` prints a short help for the module and exits.

`-proto` prints the module's function prototype as to be included in a C source file, and exits.

`-verbose` active the verbose mode that is, the print of all functions writting on the standard output `STDOUT` or on the standard error `STDERR`. Indeed, these outputs may be redirected using the shell variables `MW_STDOUT` and `MW_STDERR` (see section 2.2.1 page 16). The verbose switch allows to suppress the redirection for only the current module.

`-vers` prints the module's version number and exits.

5 System's macros

You have been already introduced to the macros by Section 1.8. A system's macro is a Bourne Shell script which uses a normalized header. All system's macros are located in the directory `$MEGAWAVE2/sys/shell`. They are used to manage MegaWave2. The following section explains each system's macro. The next section 5.2 describes a file format which is used by some macros to describe a list of MegaWave2 modules.

5.1 Macros summary

The next pages describe each system's macro; the list is in alphabetical order.

○Macro

cmw2 - The MegaWave2 Compiler

○Usage

```
cmw2  [-gnu] [-g] [-X] [-O] [-c cc_option] [-Dname[=def]] [-Uname] [-Ipathname]
      [-v] [-w] [-Ldirectory] [-llibrary] [-dep] [-pubsyslib] [-N] module
```

○Description

This command compiles a MegaWave2 module given by `module`. Since version 1.41 of this system's macro, `module` may also be a User's macro. In that case, `cmw2macro` is called. Please refer to Section 1.9 page 9 to learn more about this compilation. The module must be located into a subdirectory of `$MY_MEGAWAVE2` (plain user) or of `$MEGAWAVE2` (administrator only).

○Options

`-gnu` : Use the Gnu C compiler `gcc` instead of the standard C compiler `cc`.

`-g` : Debug flag. Cause the compiler to generate additional information needed by the symbolic debugger. This option is normally incompatible with optimization.

`-X` : XMegaWave flag. Cause the compiler to generate an interface to include the module in the XMegaWave2 software.

`-O` : Invoke the C optimizer. This option is normally incompatible with debugging.

`-c` : pass the following argument `cc_option` to the compiler.

`-Dname` : Define `name` to the preprocessor, as if by `'#define'`. If your compiler uses the ANSI mode, you should define the name `__STDC__`.

`-Uname` : Remove any initial definition of `name` in the preprocessor.

`-v` : Verbose mode. Print messages about what command is running, together with the name of temporary files. In this mode, temporary files are not deleted so you can edit them or run the commands manually.

`-w` : Suppress warning messages.

`-Ipathname` : Change the algorithm used by the preprocessor for finding include files to also search in directory `pathname`.

`-Ldirectory` : Change the algorithm used by the linker to search for the libraries. The -L option causes the compiler to search in `directory` before searching in the default locations.

`-llibrary` : Include additional library given by `library` for linking.

`-dep` : Dependencies list. Cause the compiler to generate a primary dependencies list of the module. See the macro `mwdep` for more information.

`-pubsyslib` : Link module with the PUBLIC system library (adm only).

`-N` : Do not propose to run `lint` (a C program checker/verifier) in case of compilation error.

○Macro

cmw2_all - Compile all MegaWave2 Modules

○Usage

```
cmw2_all [-gnu] [-clear] [-g] [-X] [-O] [-c cc_option] [-Dname[=def]] [-Uname]
         [-Ipathname]  [-w] [-Ldirectory] [-llibrary] [-2p] [-sp] [-dep]
         [-pubsyslib] [-force] [-v tracefile] [-N processid] src_directory
```

○Description

This command compiles all MegaWave2 modules being in the given directory `src_directory`, and recursively in all subdirectories of `src_directory`. It basically calls `cmw2` as many time as necessary. The location pointed by `src_directory` has to be a subdirectory of `$MY_MEGAWAVE2` (plain user) or of `$MEGAWAVE2` (administrator only).

○Options

`-gnu` : Use the Gnu C compiler `gcc` instead of the standard C compiler `cc`.

`-clear` : Clear the following target directories before processing the compilations: `bin`, `lib`, `obj`, `mwi` and `doc/obj` (confirmation is requested). This option allows to clean the target directories; otherwise old modules stay in those directories even if the sources have been deleted (but if the system's macro `mwrm` was used).

`-sp` : Second pass. Do not compile a module if it has been already successfully compiled. This option works only with linkers which do not set a binary containing unresolved symbols to be executable.

`-2p` : Two pass. During the first pass, all the modules are compiled. The second pass calls `cmw2_all` with `-sp`. This option allows modules which depend on other modules to be successfully compiled after the second pass. This option works only with linkers which do not set a binary containing unresolved symbols to be executable.

`-g` : Debug flag. Cause the compiler to generate additional information needed by the symbolic debugger. This option is normally incompatible with optimization.

`-X` : XMegaWave flag. Cause the compiler to generate an interface to include the modules in the XMegaWave2 software.

`-O` : Invoke the C optimizer. This option is normally incompatible with debugging.

`-c` : pass the following argument `cc_option` to the compiler.

`-Dname` : Define `name` to the preprocessor, as if by `'#define'`. If your compiler uses the ANSI mode, you should define the name `__STDC__`.

`-Uname` : Remove any initial definition of `name` in the preprocessor.

`-Ipathname` : Change the algorithm used by the preprocessor for finding include files to also search in directory `pathname`.

`-Ldirectory` : Change the algorithm used by the linker to search for the libraries. The -L option causes the compiler to search in `directory` before searching in the default locations.

`-llibrary` : Include additional library given by `library` for linking.

`-w` : Suppress warning messages.

`-dep` : Dependencies list. Cause the compiler to generate the primary dependencies list of each module. See the macro `mwdep` for more information.

`-force` : Do not ask confirmation before removing target directories.

`-v tracefile` : Verbose. Output trace of compilations in the file `tracefile` that can be viewed after the command ends.

`-pubsyslib` : Link modules with the PUBLIC system library (adm only).

`-N processid` : Internal use only.

○Macro

cmw2macro - Compile a user's macro : make it available and generate the document skeleton

○Usage

```
cmw2macro [-adm] [-ret] [-path path] [-absolute] macro
```

○Description

This command writes the document skeleton (the `.doc` file) corresponding to the macro `macro`, and it creates a symbolic link in `$MY_MEGAWAVE2/shell` (or in `$MEGAWAVE2/shell` if the option `-adm` is selected), so that you can call these macro from any location. You should not have to use directly this command, since it is called by `cmw2`, `mwmakedoc` or `mwdoclatex` when needed.

○Options

`-adm` : search `macro` in `$MEGAWAVE2` (instead of `$MY_MEGAWAVE2`.

`-ret` : return filename (with pathname) of the document skeleton.

`-path path` : search `macro` in `path` (instead of `$MY_MEGAWAVE2`.

`-absolute` : create link using absolute pathname (since cmw2macro version 2.08, default is to create link with relative pathname).

○Macro

cxmw2 - The XMegaWave2 Compiler

○Usage

```
cxmw2 [-gnu] [-g] [-O] [-Dname[=def]] [-Uname] [-Ipathname] [-w] [-Ldirectory]
      [-llibrary] modules_file
```

○Description

This command allows to compile your own version of the XMegaWave2 software. The version is defined by the modules you want to include in the software. Each module name will appear into a window panel, inside a window hierarchy defined by the group of the module. The input of `cxmw2` is an ascii file which gives the list of the modules to include together with the groups hierarchy. Please see Section 5.2 to learn more about the format of this file. In order to be included in XMegaWave2, each module must have been compiled with `cmw2` using option `-X`.

The name of the XMegaWave2 run-time program is `myxmw2`.

○Options

`-gnu` : Use the Gnu C compiler `gcc` instead of the standard C compiler `cc`.

`-g` : Debug flag. Cause the compiler to generate additional information needed by the symbolic debugger. This option is incompatible with optimization.

`-O` : Invoke the C optimizer. This option is incompatible with debugging.

`-Dname` : Define `name` to the preprocessor, as if by `'#define'`.

`-Uname` : Remove any initial definition of `name` in the preprocessor.

`-Ipathname` : Change the algorithm used by the preprocessor for finding include files to also search in directory `pathname`.

`-Ldirectory` : Change the algorithm used by the linker to search for the libraries. The -L option causes the compiler to search in `directory` before searching in the default locations.

`-llibrary` : Include additional library given by `library` for linking.

`-w` : Suppress warning messages.

○Macro

mwmakedoc - Make a new documentation for the modules (Volume 3)

○Usage

```
mwmakedoc [-c] [-h] [-N] [-g group_directory] [-nobin] [+/-index] [+/-html]
          mw2dir
```

○Description

This command creates the TEX files corresponding to the Volume Three of the MegaWave2 guides ("MegaWave2 User's Modules Library"). The documentation is written in order to keep consistency with the current MegaWave2 modules and macros put in the directory `mw2dir` (e.g. `$MEGAWAVE2`, `$MY_MEGAWAVE2` or a selected copy of such directories). The TEX files are written in the directory `mw2dir/doc`, so you need write permission in this directory. The modules, located into `mw2dir/src`, must have been compiled in order to fill the target directories `mw2dir/bin` and `mw2dir/doc` (not applicable for macros). Each module and macro must be documented that is, a `M.tex` file must be written in `mw2dir/doc` for all `M` modules and macros (see Section 7 for more information).

In order to compile the documentation, run LATEX several times (as well as other commands, such as `mwmodbibtex` and `makeindex`) as mentioned at the end of the execution of this command, or set option `-compile`.

○Options

`-c` : Compile the documentation (Volume 3) after TEX files have been created.

`-h` : Make a html (HyperText Markup Language) version of the documentation (Volume 3) using `latex2html`. Need `-c`. The generation of all html files may take a while.

`-N` : Identify the command to be not the primary process (internal use).

`-g group_directory` : Limit the scanning of modules to the given group directory (and sub-directories).

`-nobin` : Do not check consistency with the binaries (i.e. a module for which the compilation failed may be documented).

`+index` : Force the creation of an index for the modules. You normally do not need to set this option, since it is the default if the `makeidx` LATEX package is installed. If the option is set, you will need this package in order to run LATEX on the generated TEX files.

`-index` : Do not create an index for the modules, even if the `makeidx` LATEX package is installed.

`+html` : Force the addition of html code in the TEX files, to make the result of `latex2html` nicer (this command may be used to translate LATEX files to HyperText Markup Language). You normally do not need to set this option, since it is the default if the `html` and `hthtml` LATEX packages are installed. If the option is set, you will need these packages in order to run LATEX on the generated TEX files.

`-html` : Do not add html code in the TEX files, even if the `html` and `hthtml` LATEX packages are installed.

○Macro

mwarch - Machine architecture

○Usage

```
mwarch [ -s || -k ]
```

○Description

This command displays the architecture of the current host. MegaWave2 assumes that two machines with the same name returned by `mwarch` can run the same executables and can link the same objects. Therefore, you may have to call `mwarch` with the option `-s` if MegaWave2 supports several implementations (corresponding to incompatible operating systems) for the same machine.

If `mwarch` returns "unknown", you cannot run your version of MegaWave2 on this architecture.

○Options

`-s` : Add to the architecture name a suffix to identify the release of the operating system.

`-k` : Give long architecture name.

○Macro

mwcleandistrib - Clean from objects and old files the MegaWave2 Distribution (adm only)

○Usage

```
mwcleandistrib mw2_distrib_dir
```

○Description

This macro can be used to remove all unnecessary files from the MegaWave2 distribution directory named `mw2_distrib_dir` (recursively inside each subdirectory), in order to clean it. Files removed are old files generated by `emacs` (i.e. ending by `~`) and objects.

○Macro

mwcmwcheck - Check the MegaWave2 compiler (adm only)

○Usage

```
mwcmwcheck
```

○Description

This macro checks the MegaWave2 compiler `cmw2` by blind-compiling some standard modules. It is intended to check if the kernel has been correctly installed on a new architecture, before calling `cmw2_all`. It should be used by the MegaWave2 administrator only, during the installation stage.

In case of errors, the output of `cmw2` is displayed and the macro exists with non-null error code. If everything goes well, you should not experiment lot of problems by compiling the whole modules of the standard distribution.

○Macro

mwdep - Make all the secondary dependencies lists

○Usage

```
mwdep [ -adm ] [-v]
```

○Description

This command has to be called after all modules have been compiled using the option `-dep` of `cmw2` or `cmw2_all`, since it needs the primary dependencies lists of each module (files `$MY_MEGAWAVE2/doc/obj/DEPENDENCIES/*.mis`). From those files, `mwdep` generates in `$MY_MEGAWAVE2/doc/obj/DEPENDENCIES` the following files, one per module `<M>`:

- `<M>.called` lists the modules called by `<M>`;
- `<M>.calling` lists the modules calling `<M>`;
- `<M>.dep` is a TEXfile which lists the modules called by `<M>` followed by the modules calling `<M>`.

The file `<M>.dep` is included by the documentation file to constitute the "See Also" field. Therefore, run `mwdep` before `mwmakedoc`, `mwdoclatex` or `mwdocxdvi` if you want to get the right "See Also" fields in your documentation.

The file `<M>.calling` is used by `cmw2` to issue a warning message when the module `<M>` is compiled: you should re-compile also all the modules listed in `<M>.calling`.

Warning : because this command uses the output generated by the link editor, it is linker-dependent. Only a subset of linkers is supported.

○Options

`-adm` : Administrator flag. The developer's directory is `$MEGAWAVE2` instead of `$MY_MEGAWAVE2`. The user must have write permission into this directory.

`-v` : Verbose. Write more information on the standard output.

○Macro

mwdoc - Easy access to the documentation

○Usage

```
mwdoc [<name> || M || S || F || 1 || 2 || 3]
```

○Description

This command offers an easy access to the documentation. Since objects doc files are in the DeVice Independent (DVI) file format, your system must support the DVI previewer `xdvi`. If you call `mwdoc` without argument, it will ask you to tell him what kind of doc you are seeking. You may also put directly in argument the doc you want.

○Options

`<name>` : The word `<name>` is supposed to be the name of a module or of a user's macro. This causes to view the doc of the correponding module or user's macro.

`M` or `m` : List all modules and user's macro with a short description.

`S` or `s` : List all system's macros with a short description.

`F` or `f` : List all available external (file) types with a short description. To reduce the list to the file types you are looking for, after the prompt specify a keyword.

`1` : View the Volume 1, MegaWave2 User's Guide.

`2` : View the Volume 2, MegaWave2 System Library.

`3` : View the Volume 3, MegaWave2 User's Modules and Macros Library.

○Macro

mwecho - Portable echo with -n and -E options

○Usage

```
mwecho [-n] [-E] ...
```

○Description

This command emulates a portable `echo` command with -n and -E options. The command `mwecho -n` replaces `mwechon` which is no more furnished.

○Options

`-n` : Do not output the trailing newline.

`-E` : Disable interpretation of backslash-escaped characters.

○Macro

mwinstall - Install MegaWave2 (administrator only)

○Usage

```
mwinstall [-static] [-public || -public=private] [-clear]
          [-debug] [-level l] mw2distribution
```

○Description

This command is for the MegaWave2 Administrator only. Remember that the Administrator is the one which is supposed to install, maintain and update MegaWave2 for all users. This command may be called directly, but you may also use the shell `Install` in the root directory of `mw2distribution` where `mw2distribution` is an original MegaWave2 Distribution Package (`$PRIVATE_MEGAWAVE2`). The main steps of the installation are as follows :

1. The environment variables needed to run most macros are built, using the macro `mwsetenv`.
2. The kernel is compiled for your machine architecture, using installation shells and makefiles located in `mw2distribution/kernel`. The kernel is composed by the Wdevice library (interface with the Window System), by the System library, and by the preprocessor.
3. The modules and user's macros are compiled for your machine architecture, using the macro `cmw2_all`.
4. The volume 3 of the documentation (User's Modules and Macros Library) is generated, using macros `mwdep` and `mwmakedoc`.
5. If the option `-public` has been selected, a list of successfully compiled modules and user's macros is written using the macro `mwmodlist`, and modules and user's macros of this list are installed into the public MegaWave2 using the macro `mwmodinstall`.

The macro `mwinstall` is intended to allow an easy installation process on the very most common machines and configurations only. It means that something goes wrong is not an unlikely event. In that case, you will have to correct the wrong things by yourself and to rerun the macro, eventually using the option `-level l` to avoid levels already completed. Or you will have to make the installation "manually" that is, by calling lower-levels macros. Such manual installation is not a bad idea, even if everything goes well with `mwinstall`. A manual installation allows to customize MegaWave2 more deeply than the standard one. For example, you may want to install in the public MegaWave2 a subset of the private modules only. This can be easily done by editing the list of modules written by `mwmodlist`, before calling `mwmodinstall`.

Be aware that, since the installation is machine-dependent, you should install MegaWave2 on each machine architecture for which you want to be able to use it. Of course, if for each

installation you specify the same public MegaWave2 directory, source files will not be duplicated.

○Options

`-static` : Make static kernel libraries (by default, kernel libraries are shared). Selecting this option may be hazardous since modules linked with static libraries are of big size. You should use it after having installed the shared libraries only (do not select `-clear` or you will remove them), so that by default modules would be still dynamically linked. To statically link a module, call `cmw2 -c cc_opt` with `cc_opt` the corresponding cc option (e.g. `-c -static` with `gcc`). The advantage of a module statically linked lies in the fact that it can be executed without the libraries. In addition, some debuggers have strange behavior when used with dynamically linked modules.

`-public` : Install two MegaWave2, the private one for the administrator only and the second one for plain users. In this way, the administrator will be able to change and check modules, user's or system's macros and functions of the kernel libraries without touching the public version. This option is obviously usefull only if several persons intensively use MegaWave2, and if you plan to update MegaWave2.

`-public=private` : Allow other persons than the administrator to use the private MegaWave2. By default, the administrator is the only one supposed to use the private MegaWave2.

`-clear` : Clear objects in the previous installation.

`-debug` : Pass the debug option to the C compiler. By default, the C compiler is called with the optimization flag.

`-level l` : Do not start the installation from the beginning (level 1) but from level l. This supposes that previous levels have been successfully completed. In particular, the environment variables must be correctly set in the shell from which you call `mwinstall`.

○Macro

mwmodbibtex - Run BibTeX on the MegaWave2 documentation (guide #3) and add reference to modules

○Usage

```
mwmodbibtex mw2dir
```

○Description

This command replaces the `bibtex` command included in standard TEX packages, when compiling the "MegaWave2 User's Modules Library" documentation (guide #3) into the subdirectory `mw2dir`/doc. In addition of calling `bibtex` to create a bibliography file `guid3.bbl`, `mwmodbibtex` adds to the bibliography file a list of references to modules : at the end of each bibliographical reference, the list of modules making citation to this reference is added. As a result of, the guide #3 obtained by normally post-processing the file with LATEX (as written when the command `mwmakedoc` ends) will contain, in the bibliography section, a list of modules associated to bibliographical references.

As well as `bibtex`, use of `mwmodbibtex` assumes at least one bibliographic database (.bib file) exists. The needed one is `$MEGAWAVE2/doc/public.bib` which contains references for public modules. It is normally included in standard MegaWave2 distribution. Put references for private modules in `$MY_MEGAWAVE2/doc/private.bib`.

In addition to standard fields one uses into public.bib and private.bib files, one may specify a WWW url by using the special command `\hturl`, as in the following :

```
  note   =  "See \hturl{http://www.cmla.ens-cachan.fr/Cmla/Megawave}"
```

By doing this, the corresponding url will become an active hyperlink when the LATEX documentation file will be processed through the `latex2html` command (and no error will be encountered if the `latex2html` package is not available).

Beware : when documenting module M in M.tex, do not collapse citations.
Use e.g. `\cite{key1}\cite{key2}` instead of `\cite{key1,key2}`.

○Macro

mwmodinstall - Install a new MegaWave2 public modules environment (administrator only)

○Usage

```
mwmodinstall [-clear] [-tdir target_directory]
             [-X file_of_xmw2_modules] file_of_modules
```

○Description

This command allows to copy selected modules and user's macros from the source directory `$PRIVATE_MEGAWAVE2` to the target directory `$PUBLIC_MEGAWAVE2`. It can be used by the administrator only to make some private modules available for all. The command copies not only the module sources, but also everything associated such as objects, binaries, documentation and data. The content of `$PRIVATE_MEGAWAVE2/data/PUBLIC` is always entirely copied.

The file `file_of_modules` lists the modules to be copied. See Section 5.2 for more informations about such modules file. In the case where your configuration is such that `$PRIVATE_MEGAWAVE2 = $PUBLIC_MEGAWAVE2`, this macro is obviously useless.

Before running this command (which may take a while), run `mwmodcheck` to check the consistency of the modules file regarding the content of the source directory. This macro will tell you any modules listed in the file but not in the source directory, incomplete modules (e.g. without documentation attached) and, for information, modules which will remain private (i.e. in the source directory but not listed in the file).

○Options

`-clear` : Clear previous modules in `$PUBLIC_MEGAWAVE2`. This option should be used each time the administrator wants to make a complete new version of the public modules.

`-tdir target_directory` : Change default target directory `$PUBLIC_MEGAWAVE2` to be `target_directory`.

`-X file_of_xmw2_modules` : Generate a public XMegaWave2 software containing the modules listed in the file `file_of_xmw2_modules`.

○Macro

mwmodlist - List the modules and macros found in a MegaWave2 source directory

○Usage

```
mwmodlist [-mfile] [-bad] [-group gdir] mw2dir
```

○Description

This command prints all the modules and macros found in the directory `mw2dir`, which has to be a MegaWave2 directory such as `$MEGAWAVE2` or `$MY_MEGAWAVE2`. In case of a module not successfully compiled, a warning is issued.

○Options

`-mfile` : print the list in the modules file format (which is compatible with input of macros like `mwmodinstall`). See Section 5.2 for more informations about this format.

`-bad` : print modules which are not successfully compiled only.

`-group gdir` : restrict the list to the group and subgroups of `gdir`.

○Macro

mwmodsearch - Search for modules matching words

○Usage

```
mwmodsearch [-l] [-public] [-private] word_1 [word_2 ... word_n]
```

○Description

This command searches for modules and user's macros matching the given list of words. It is useful when you are seeking a specific algorithm without any idea about the module's name or even about the group's name, and when the short function description (as returned by `mwmodlist`) is not enough to perform a search on.

With `mwmodsearch`, the search is performed both on the source file and on the documentation (tex file). Only one matching file is enough to satisfy the search. But when several words are given, all words must be found in the same file.

By default, each matching lines are printed together with the name of the file. This may result to a pretty huge number of lines if the words are common. In such a case, use -l and try to reduce the number of matching modules by adding new keywords.

○Options

`-l` : list only. Do not output matching lines, but only the list of matching modules.

`-public` : restricts search to public modules and user's macros.

`-private` : restricts search to private modules and user's macros.

○Macro

mwnewuser - Create the directory hierarchy for a new user

○Usage

```
mwnewuser
```

○Description

If you are a new user, you may run this command once: it creates the directory `$MY_MEGAWAVE2` and all subdirectories needed by the MegaWave2 system's macros.

○Macro

mwrm - Remove module(s) or user's macro(s)

○Usage

```
mwrm [-macro] M
```

○Description

This command searches all modules (or user's macros) named `M` and removes them from the system (confirmation is requested before to operate). By using this command instead of directly removing the source files, you make sure to remove all objects and references attached to the modules.

Only the MegaWave2 administrator should be able to remove public modules and macros. Therefore make sure the `$PUBLIC_MEGAWAVE2` directory is not writable for plain users.

○Options

`-macro` : Say that `M` is a user's macro and not a module. You normally don't need to set this option, since the command recognizes if `M` is a macro by its name, which should begin with a capital letter.

○Macro

mwrnwordmod - Rename words inside modules

○Usage

```
mwrnwordmod [-adm] [-all] [-check] [-d dir] [-f file_of_names]
```

○Description

Sometimes the name of a system's function, of a structure or of a structure field may change. In such a case, you may use this macro in order to automatically perform the change of name inside each modules.

At the location of the macro `mwrnwordmod`, you can also find the file `mwrnwordmod.data` : this is an Ascii file which gives the list of words to change, in the obvious format `old_name new_name` (one change per line).

This list is updated to reflect the last change of names we have performed in the kernel. By default (if you do not use the `-f` option), this is this file which is read to perform the changes. The `mwrnwordmod.data` is usefull if you have written lot of modules and if you install a new MegaWave2 kernel, for which some function names or structures have changed.

Warning :

- The replacement is performed using the stream editor `sed`. Therefore, the string(s) to be searched is specified by a `regex` (regular expression).
- Be aware that the content of the file is not analyzed : for example, if you wish to change a field named `open` to `is_open`, all occurrences of "open" will be touched, including those inside comments.
- It is recommended to make a copy of the `src` directory before running this macro : if the result does not match your wish, you will be able to restore the old state.

○Options

`-adm` : administrator flag. Scan `$MEGAWAVE2/src` directory instead of the default `$MY_MEGAWAVE2/src`

`-all` : scan all files (not only modules, that is .c files)

`-check` : check only. See files that would be changed, but change nothing.

`-d dir` : scan directory `dir` instead of `$MY_MEGAWAVE2/src` or `$MEGAWAVE2/src`.

`-f file_of_names` : use this file instead of the list given by `mwrnwordmod.data`.

○Macro

mwsetenv - Set up the environment variables needed by MegaWave2 (adm only)

○Usage

```
mwsetenv [-public=private || privateonly] mw2distribution
```

○Description

This command helps the administrator to set the environment variables needed by most user's macros. See Section 2.1.4 to learn more about the environment variables. The directory `mw2distribution` must corresponds to an original MegaWave2 Distribution Package, its name will be the value set to the variable `$PRIVATE_MEGAWAVE2`.

As a result of, this macro write two files to be used by the administrator and two files for plain users (one for Bourne-compatible shells and one for C-compatible shells). Those files, which may have to be customized, are to be included in the `.profile` or the `.cshrc` file. The location of these files are given at the end of the macro execution.

○Options

`-privateonly` : Select this option if there is no public MegaWave2. By default, this macro considers that both a public MegaWave2 and a private one has to be installed.

`-public=private` : Select this option if there is no public MegaWave2 and if the installation allows other persons than the administrator to use the private MegaWave2.

○Macro

mwsysmaclist - List all system's macros

○Usage

```
mwsysmaclist
```

○Description

This command prints all the system's macros found in the directory `$MEGAWAVE2/sys/shell`.

○Macro

mwvers - MegaWave2 Version

○Usage

```
mwvers [-major || -minor || -variant]
```

○Description

This command returns the current version number of the public MegaWave2 software you are using. Please refer to this version number when you report bugs. On MegaWave2 Version 1.x, this command was available on MegaWave2 system issued from the distribution package only. This is no more the case.

○Options

`-major` : Return the major version number only (e.g. 2 if the full version is 2.00a.12)

`-minor` : Return the minor version number only (e.g. 00a if the full version is 2.00a.12)

`-variant` : Return the variant version number only (e.g. 12 if the full version is 2.00a.12)

○Macro

mwwhere - Give the location of the source of a module or user's macro

○Usage

```
mwwhere [-macro] [-bin] M
```

○Description

This command returns the path where the source of the given module or user's macro `M` is found, searching first for private modules and then for public ones.

○Options

`-macro` : Say that `M` is a user's macro and not a module. You normally don't need to set this option, since the command recognizes if `M` is a macro by its name, which should begin with a capital letter.

`-bin` : Return the path if the corresponding binary (executable) exists only. When two paths are returned with this option set, it means that one module is hidden (with default settings, the private module hides the public one). In such a case, the macro exits with value 2.

○Macro

mwdoclatex - Make the documentation of a module or of a user's macro

○Usage

```
mwdoclatex [-adm] M
```

○Description

This command compiles a single module documentation file using LaTeX. Use it instead of the macro `mwmakedoc` when you want to print the documentation of only one module or user's macro (for example the one you just finished to write). If you rather want to see the documentation on the screen, use the macro `mwdocxdvi`.

Run `mwdoclatex` after `cmw2` has been called so that the document skeleton `M.doc` exists. Make sure to have written the corresponding TeXfile (`M.tex`), see Section 7.

This command calls LaTeX and, as the result, you get a `M.dvi` file into the `$MY_MEGAWAVE2/doc/obj` directory which can be used to print the documentation.

○Options

`-adm` : Say that `M` is a public module or a public macro from `$MEGAWAVE2` and not from `$MY_MEGAWAVE2`. In that case, if you are not the administrator but a plain user, and if the administrator didn't make this doc already, the dvi file `M.dvi` is created into `$MY_MEGAWAVE2/tmp/megawave2_doc/user`.

○Macro

mwdocxdvi - Display on the screen the documentation of a given module

○Usage

```
mwdocxdvi [-adm] [-macro] M
```

○Description

This command compiles a single module documentation file using the macro `mwdoclatex`, and then call the DVI previewer for the X Window System, `xdvi`, to display the documentation on the screen.

○Options

`-adm` : Say that `M` is a public module or a public macro from `$MEGAWAVE2` and not from `$MY_MEGAWAVE2`. In that case, if you are not the administrator but a plain user, and if the administrator didn't make this doc already, the dvi file `M.dvi` is created into `$MY_MEGAWAVE2/tmp/megawave2_doc/user`.

`-macro` : Say that `M` is a user's macro and not a module. In that case, the macro `cmw2macro` is called to generate a file `M.doc`. You normally don't need to set this option, since the command recognizes if `M` is a macro by its name, which should begin with a capital letter.

5.2 List of modules

Some system's macros (e.g. `cxmw2`, `mwmodinstall`) need to know a list of modules you want to process together with the group hierarchy. MegaWave2 uses a plain ascii format to describe such a list in a file. You can write this file manually, or you can use the output generated by the macro `mwmodlist` using `-mfile` option.

Each line of the file is normally filled by a module name. They are special symbols which change the meaning of the line (those have to be the first character of the line)

- `%` : comments. The line is ignored.
- `#` : keyword. The following keywords are available:
  - `#group group_name` : Give the group to which the next modules (or subgroups) belong. The hierarchy of the groups must be given that is, if `B` is a subgroup of the main group `A`, the line `#group A` must appear and after that, the line `#group A/B`.
  - `#dir dirname` : Change the default source directory for the modules to be `dirname` (default is `$MEGAWAVE2`). Do not use this keyword for `cxmw2`.

6 User's macros

You have been already introduced to the macros by the section 1.8. An user's macro is a Bourne shell script which uses a normalized header, and which calls a sequence of commands, such as MegaWave2 modules.

Since system's macros use the header of the user's macro to perform some operations, as to make the documentation of the macro (see section 7.2), you have to correctly fill the header when you write a new macro.

In order to easily recognize a user's macro from a user's module (they are located in the same source directory), we use capital letters to name macros (at least for the first letter) and lower letters to name modules. We recommand you to follow this convention.

6.1 Header of a Macro

Each macro must have an header which consists of variables to be filled. Falling to define most of these variables leads malfunctions when calling some systems's macros.

In the following is the list of the variables to define

- `_Prog`: Name of the macro.
- `_Group`: Group where belongs the macro.
- `_Func`: Short description of the function performed by the macro.
- `_Vers`: Version number of the macro.
- `_Date`: Year of the creation or last change.
- `_Auth`: Name of the author.
- `_Usage`: Usage line, following Unix command conventions (do not mention the program name).
- `_Labo`: Name and address of the institution, when it is not the CMLA.

You can get examples of such headers by reading the source of some systems's macros (which share the same header conventions) or of some public user's macros given in the distribution.

6.2 How to use user's macros

An user's macro being a Bourne shell script, you don't really need to compile it before to call it. But in order to be able to call it from any location, a link has to be created in a directory recorded in the path variable (which is `$MEGAWAVE2/shell` if you are the administrator, `$MY_MEGAWAVE2/shell` otherwise). In addition to that, the documentation must also be generated. Therefore, you should compile a user's macro as if it was a module, using `cmw2`.

7 Documentation

Warning : in MegaWave2 versions 2.x, the `doc` directory structure has changed. Now put the documentation you write (the `.tex` files) into the `$MEGAWAVE2/doc/src/` directory (administrator) or `$MY_MEGAWAVE2/doc/src/` (plain user) directory, without creating subdirectories for the groups : the `.tex` files of all modules of all groups belong to the same directory. The `$MEGAWAVE2/doc/obj/` and `$MY_MEGAWAVE2/doc/obj/` directories contain files that are automatically generated (such as `.doc` and `.dvi` files). Do not put your `.tex` files here, they may be destroyed by the system !

7.1 Document a module

Each module you write must be documented. MegaWave2 helps you as much as possible in this unattractive task: when you compile a module, a document skeleton is generated in the directory `$MEGAWAVE2/doc/obj/` (or `$MY_MEGAWAVE2/doc/obj/`). It takes the name of the module and the extension `.doc`. This file documents in the LaTeX language about everything that could be automatically done, as the synopsis of the run-time command, the summary of the module function, the release number and the copyright for the authors and laboratories.

(Un)Fortunately, there will always be a part of the documentation which cannot be automatically generated: the mathematical description of the algorithm. This part is the *Description* field of the documentation. Therefore, the author of a new module must write this content in a file into the `$MEGAWAVE2/doc/src/` (administrator only) or `$MY_MEGAWAVE2/doc/src/` (plain user) directory. Give this file the name of the module with the extension `.tex`. As the file will be inserted by MegaWave2 into the whole documentation, you must write the text using a subset of the LaTeX language. In particular, don't use any commands about the style or the presentation (as `\begin{document}`, `\documentstyle`, `\newpage`, `\section`, ...).

You can get a lot of examples of such documents by seeking the files into subdirectories of `$MEGAWAVE2/doc/src`.

Since MegaWave2 V 2.21, the system uses bibliographic databases where all citations are recorded. As this allows cross-references, it is now possible to get all modules associated to a given article. In order to fulfill the new requirements, citations have to be set in the `.tex` file using the standard `\cite` LaTeX command, and the corresponding references have to be given in the bibliographic database (.bib file), NO MORE in the `.tex` file. Beware, do not collapse citations : use e.g. `\cite{key1}\cite{key2}` instead of `\cite{key1,key2}`. References for private modules have to be put in `$MY_MEGAWAVE2/doc/private.bib` while references for public modules are in `$MEGAWAVE2/doc/public.bib`. Of course, only references for private modules not in `public.bib` have to be in `private.bib`. See the system macro `mwmodbibtex` for more information about the bibliographic database.

7.2 Document a macro

The documentation of a macro follows the same rules as the ones for modules. Therefore, for each new user's macro, you must write a description file `$MEGAWAVE2/doc/src/MACRO.tex`

(for a public macro, administrator only), or `$MY_MEGAWAVE2/doc/src/MACRO.tex` (for a private macro), `MACRO` being the name of the macro located in a subdirectory of `$MEGAWAVE2/src/` (or of `$MY_MEGAWAVE2/src/`). In this file, you put in LaTeX the part which corresponds to the *Description* field of the documentation.

The difference between macros and modules is that macros do not need really need a compilation, and therefore you may call a specific command to generate the document skeleton `$MEGAWAVE2/doc/obj/MACRO.doc` (or `$MY_MEGAWAVE2/doc/obj/MACRO.doc`), which is the macro `cmw2macro` (see section 5.1). Since MegaWave2 Versions 2.x, `cmw2` will also perform this task if its argument is a user's macro.

7.3 Print the documentation

Once the document skeleton of a module or macro is generated, and the description file is written, you can compile the document skeleton by calling the macro `mwdoclatex` (see section 5.1). Of course this needs to have the LaTeX environment installed on your system. You get a `DVI` file which can be printed on various devices using commands of the standard TeXware distribution.

If you want to print the documentation of all modules and user's macros instead of only one, use the macro `mwmakedoc`. This will actually create an updated sample of the Volume Three of the MegaWave2 guides ("MegaWave2 User's Modules Library").

In the document skeleton of each module (not available for macros), a field named "See Also" lists all the modules calling the module or called by the module: this is a dependencies list. In order to update this list, you need to compile each module which the option `-dep` and you need to run the macro `mwdep` before the macros `mwdoclatex` and `mwmakedoc`.

8 Annex

8.1 License

You do not have to send a registration form any longer, but by using MegaWave2 you are still supposed to follow the term of the MegaWave2 Public License below.

MEGAWAVE2 PUBLIC LICENSE
Terms and conditions for using, copying, distribution and modification of MegaWave2 version 2.x.

You acknowledge to be informed about the following facts, and you accept the consequences:

- MegaWave2 is a "soft-publication" for the scientific community, maintained by the image department of the CMLA, URA CNRS 1611 de l'Ecole Normale Supérieure de Cachan.
- The image department is founded by the CMLA only, using public funds and private grants for software development.
- MegaWave2 has been implemented for research purposes only; it comes therefore without any warranty.
- The MegaWave2 compiler and the system libraries are known to run on some computer models and operating systems only.
- The CMLA may not maintain this software in the future.
- The CMLA can not help any user to install nor to use the software.
- The CMLA may not publish all the code contained in the various versions of MegaWave2.
- For transparency, the published code contains the source of the mathematical algorithms developed at the CMLA. You are allowed to copy, modify and redistribute these programs.

The CMLA claims to impose on the diffusion and on the use of its software the same rules as the well-known rules which apply to paper publication.

You agree to

- inform without delay the CMLA (see address next page) about any contract between yourself or your department and any private or public organization, whenever its execution uses an algorithm (modified or not) included in MegaWave2. You will inform this organization about the origin of these algorithms by writing in the contract the use of the MegaWave2 software.
- mention in all products (as computer programs or scientific papers) which take advantage of an algorithm included in MegaWave2 (modified or not), the use of this software together with the name(s) of the author(s) written in the header of the corresponding module and with the address of the CMLA and of other laboratories implied in the development, if any.

8.2 How to contact the CMLA

Please visit our *World Wide WEB* homepage at the following URL :
`http://www.cmla.ens-cachan.fr/Cmla`. By selecting the *MegaWave* link, you will get the latest news on MegaWave2.

At this site you will be able to

1. download the latest version of MegaWave2;
2. get access to the MegaWave2 forum (to submit a question and to read the answers given to former questions);
3. submit new modules;
4. propose some corrections on the existing modules and kernel;
5. inform the CMLA about contracts or products which use algorithms included in MegaWave2, as it is required by the MegaWave2 Public License;
6. report bugs, remarks, ...

Be sure that, even if we are not always able to answer, all mails will be read and remarks will be take into consideration for a new release.

Address: MegaWave, CMLA, Ecole Normale Supérieure de Cachan, 61 avenue du Président Wilson, 94235 Cachan cedex, France.

E-mail: megawave@cmla.ens-cachan.fr

Index