Notes on Mk4 software directory structures and utility routines

Modified by CJL, 31 March 1999

There is a pair of shell scripts in /correlator/prog which, when executed, should set up the various environment variables referred to in my document of March 23 on the directory structure. The shell scripts are called **mk4_setup.csh** and **mk4_setup.sh** for csh (and derivatives) and sh (and derivatives) respectively.

Before these scripts are executed, the ARCH environment variable must be set (currently only "linux" and "hppa" are supported).

In the UTIL library ($UTIL/$ARCH/libutil.a) there is a routine called **environment()**, which initializes several external character strings with corresponding environment variable settings. If an environment variable is undefined, the string is set to a default value by **environment()**. There are no arguments. To use this routine, simply call it at the start of your program. By declaring the strings as externs, they can be accessed in any subroutine. Their names, and the environment variables they correspond to are:

datadir	DATADIR
scheddir	SCHEDDIR
afiledir	AFILEDIR
textdir	TEXT
sysvexdir	SYSVEX
taskdir	TASK
bindir	BIN
tmpdir	TMP

so if you need file "xyz" in the sysvex directory, you might use code something like this:

```
...
...
main()
...
...
environment();
...
...
void thisfunction()
   {
   extern char sysvexdir[];
   char rel_filename[] = "xyz";

   sprintf (filename, "%s/%s", sysvexdir, rel_filename);
...
```

I have constructed the routine **emsg()** discussed by Roger. The logical place for it is in $MESS (=$CORRSUB/mess), which will also contain mess.c. The **emsg()** call is used only by the online Mk4 system, so belongs in the $CORR tree, not in $UTIL. A small makefile will be needed to build a libmess.a library archive for linking, containing emsg.o and mess.o. The source code only is currently in $MESS, pending JAB's transferral and suitable modification of mess.c and mess.h.

The routine has a variable argument list just like **printf()**, is called in the same way, and does the same formatting as **printf**. There are, however, two additional arguments immediately after the format string, namely **level** and **errnum**.

The first is an importance level argument for verbosity control. There is an extern int called **error_level**, declared in **emsg.c**, which defaults to 0. If the **level** argument in the call to **emsg()** equals or exceeds **error_level**, the error message gets sent to the appropriate target program or programs (see below), otherwise nothing happens. Normally, various fatal and non-fatal error conditions would be coded with different **level** argument values. The value of the **error_level** extern can be manipulated by a command line flag, a message sent from another program, or both, in order to control the verbosity of the error message stream generated by the program.

The second extra argument is composed of two parts. The first is a unique error number, which acts as a key into a file of descriptive error messages of arbitrary length. Defines for these error numbers will be provided in the header file $INC/mk4_errno.h. Also in this header file are **#define** statements for bits corresponding to various programs in the online system. These bits can be OR'ed into the **emsg** argument in order to specify the destination(s) for the error message. If no destination bits are set, **emsg** defaults to sending the message to opera.

The routine prepends the name of the program to each message. This name is contained in the extern character string **emsg_progname**, which has a frivolous default. Programs which use **emsg** should reset this extern to a sensible value upon startup.

A typical call might be:

```
#include "mk4_errno.h"

        if (error)
                {
                emsg ("Tape drive number %d has exploded - check for casualties",
                                                        2, BANG | CONDUCTOR, driveno);
                return (-1);
                }
```