FlagShip

Object Oriented Database Development System

Cross-Compatible to Unix, Linux and MS-Windows

MULTISOFT

Release 8.1

Section SYS

The whole FlagShip 8 manual consist of following sections:

Section	Content
GEN	General information: License agreement & warranty, installation and de-installation, registration and support
LNG	FlagShip language: Specification, database, files, language elements, multiuser, multitasking, FlagShip extensions and differences
FSC	Compiler & Tools: Compiling, linking, libraries, make, run-time requirements, debugging, tools and utilities
CMD	Commands and statements: Alphabetical reference of FlagShip commands, declarators and statements
FUN	Standard functions: Alphabetical reference of FlagShip functions
OBJ	Objects and classes: Standard classes for Get, Tbrowse, Error, Application, GUI, as well as other standard classes
RDD	Replaceable Database Drivers
EXT	C-API: FlagShip connection to the C language, Extend C System, Inline C programs, Open C API, Modifying the intermediate C code
FS2	Alphabetical reference of FS2 Toolbox functions
QRF	Quick reference: Overview of commands, functions and environment
PRE	Preprocessor, includes, directives
SYS	System info, porting: System differences to DOS, porting hints, data transfer, terminals and mapping, distributable files
REL	Release notes: Operating system dependent information, predefined terminals
APP	Appendix: Inkey values, control keys, ASCII-ISO table, error codes, dBase and FoxPro notes, forms
IDX	Index of all sections
fsman	The on-line manual “f**sman**” contains all above sections, search function, and additionally last changes and extensions

multisoft Datentechnik, Germany

Object Oriented Database Development System,
Cross-Compatible to Unix, Linux and MS-Windows

Section SYS

Manual release: 8.1

For the current program release see your Activation Card,
or check on-line by issuing *FlagShip -version*

Note: the on-line manual is updated more frequently.

Copyright

Made in Germany. Printed in Germany.

Trademarks

Headquarter Address

multisoft Datentechnik
Schönaustr. 7
84036 Landshut
Germany

Phone: *(*+49) 0871-3300237

E-mail: support@flagship.de
support@multisoft.de
sales@multisoft.de

Web: http://www.fship.com

SYS: System and Porting Hints

1. Porting Hints

FlagShip is designed as a stand-alone development system in the Unix/Linux or MS-Windows environment in order to create highly efficient database applications with little effort on the part of the programmer.

Because of its highest cross compatibility on the market, it is also ideally suited for porting existing applications written in Clipper, FoxBase, FoxPro, dBASE or other xBASE languages to nearly any Unix operating system.

Although the Unix system is quite different from MS-DOS, we at multisoft make every effort to make your program porting as easy as possible. Here are some hints on programming techniques, which can enable you to write your sources fully portable, almost without any modifications.

1.1 System differences Unix and DOS

Unix is designed as a general purpose, multiuser and multitasking operating system running on nearly any computer or processor, from a low- cost computer to mainframe. In contrast, MS-DOS is a single-user, single- tasking operating system running on Intel based processors only.

On Unix, you do not need additional hardware and software (such as a network) to allow several users to access the same data. By using virtual terminals or background processing, different tasks (applications) may run simultaneously on the same terminal. Additional terminals may be connected via serial cabling, modem, Ethernet and so on. A normal PC, running a terminal emulator program, is also often used as a terminal for Unix.

The main difference between Unix and a networked PC is where the application is executed and how it handles the data requested. On Unix, the application always runs on the main Unix computer (server). Distributed processing, by sharing the data and applications is also possible using the TCP/IP protocol, NFS or other remote networks.

Code sharing

When the same application is used by different users, the executable code is usually loaded only once into the RAM memory and shared with all other users. Of course, any user (process) receives its own, protected data storage. Code sharing can usually be controlled by means of linker "ld" options (-i -A -F on SCO, see your "man" pages) and is set by default when compiling by FlagShip. Not available in DOS, partially available in Windows and supported as such automatically in FlagShip for MS-Windows.

Mathematical coprocessor

If a mathematical coprocessor is detected (standard nowadays), FlagShip uses it for all floating point operations (i.e. with all numeric variables) and during internal processing. This speeds up the application execution significantly, compared to the same computer without a coprocessor.

Memory swapping and paging

Most current Unix systems (system V [five] release 3 and up) allow running a single application or the sum of all user applications to be larger than the available RAM storage – at least four Gigabytes or more. If the RAM storage is insufficient, the Unix kernel swaps a part of the currently inactive RAM storage to disk and loads another program part into RAM. Some of the swapping parameters can be changed by the user (system manager, super-user) by tuning the kernel parameters. See details in chapter SYS.3.

MS-Windows 32/64bit supports paging and swapping by a similar way as in Unix, using a swapping file.

On DOS, the application can use only up to 640 KB. The internal memory swapping is very limited and must be handled by the application itself, e.g. using overlays, swapping handles, extended or expanded RAM memory handles and so on.

Data access

On DOS, the application always runs exclusively on one computer. When data sharing with other users is required, additional network hardware and software must be used. The network simulates additional disk storage to the local PC. When accessing the (remote) disk, all the data is shuffled through the network to the local PC until the required record is found.

In Unix, the application runs on the same hardware where the data is located. This means that additional data transfer is not required and the throughput is much higher as compared to a network. (Similar to running an application directly on the network server). The Unix system has additional internal data buffering (caching), which is also tunable. See chapter SYS.3.

In MS-Windows, the application can run either on the workstation, or on the server, see also Networking below.

Data sharing management

When data sharing is required, both Unix and DOS network support similar mechanisms. In order for several users to share the same file as read-only (like reading a text file), there are no special requirements.

When a read-write access is involved (as database or indices are usually used), several conditions must be met to avoid mutual, uncontrolled overwriting of the same data. Firstly, the file must be opened in shareable mode. Prior to any write attempt, the data (record or file) must be "locked". This marks a part of the file as "in use by my application", if not already in use by another task. Moreover, the executed program must ensure that a write access without previous locking is rejected. All these mechanisms are set automatically by FlagShip simply by using the SHARED clause of the USE command (or SET EXCLUSIVE OFF) and the RLOCK(), FLOCK() ... UNLOCK locking mechanisms for databases. If SET AUTOLOCK is ON (the default), FlagShip automatically locks the record or file for the duration of replacement and unlocks it thereafter. Also, low-level locking is available using the FLOCKF() function.

The number of shareable files (per user and system-wide), and the number of locks available is controlled by the Unix kernel and its tunable parameters (see SYS.3). The currently set locks may be examined using e.g. the Unix command "sar -r 1 1" or in Linux by “cat /proc/locks”.

On DOS, the number of shareable files is set in different parts: in the CONFIG.SYS file with FILES=, by loading the network software, using the SHARE command and by the environment variable CLIPPER=Fxxx. The number of locks is not configurable. In 32 and 64bit Windows and Unix/Linux, there is no need such a setting for executables created by FlagShip.

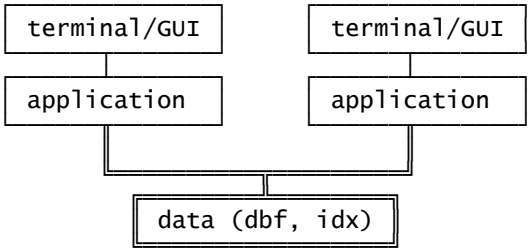
File system, drive letters

FlagShip for MS-Windows automatically supports DOS drive selectors. There is no Unix equivalent for the MS-DOS drive selector, e.g. `A:` or `D:`. Instead, Unix utilizes the concept of physically mounting separate disks into one file system using a tree-like structure of directories. FlagShip for Unix/Linux automatically supports DOS drive selectors specified in the source, when the corresponding environment variable is set (e.g. "`export D_FSDRIVE=/home/data`" to automatically substitute /home/data as drive `D:`). See section FSC.3.3 for details.

Networking

The data (databases, indices) may be located on a local disk drive, or on a remote file system (network) mounted via NFS, SAMBA, Windows server, Novell, WLAN/Intranet/Internet etc. You will achieve best performance when the application access local data; network access may slow-down the performance (sometimes significantly, in dependence on the used topology and LAN speed).

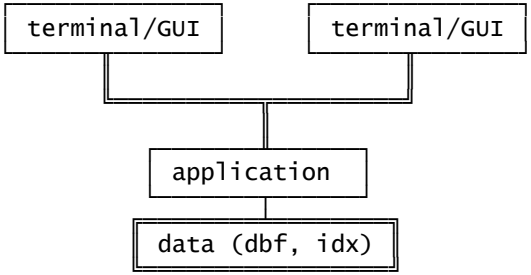
FlagShip supports different network topology:



Workstations/Users on the same or different operating system(s)

Network (LAN, WLAN) supporting byte locking

Databases and indices on server



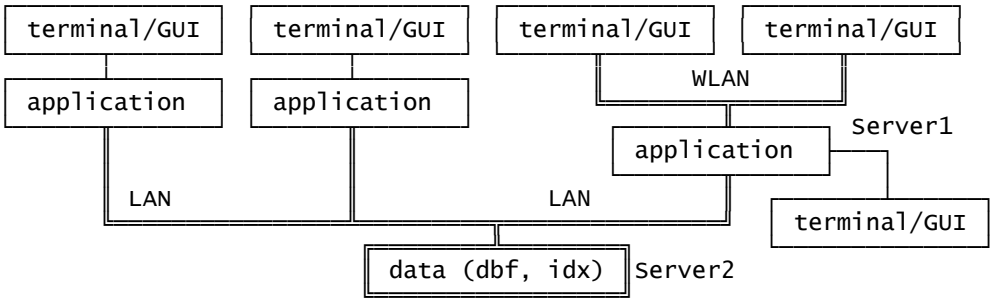
Workstations/Clients remotely connected to server (telnet, ssh, emulator, X/server etc.)

Any network (LAN, WLAN)

Application on server (aka client/server)

Databases and indices on server

or in any combination of above, e.g.



If the application and data are at different physical locations, the network protocol (or at least the network part connecting application to data) must support byte locking. This is required to handle file FLOCK() and record RLOCK() locking in FlagShip. Here some common settings for configurable networks:

NFS: (Unix/Linux/Windows client, settings usually in /etc/fstab file) server:/path /mountpoint nfs ...rw,exec,lock,sync,....

```
example:
192.168.1.1:/data /server/data nfs user,rw,exec,auto,lock,nfsvers=3,
                    sync,hard,intr,rsize=8192,wsize=8192 0 0
```

Note: on instable NFS WLAN network, you may need to invoke SET NFS ON in your application to avoid data corruption, see section CMD for details.

SAMBA: (server for Windows, settings usually in /etc/samba/smb.conf file)

```
[share-name]
    locking = yes          # just to be sure, default=yes
    strict locking = no    # yes would always lock the whole file
    oplocks = no           # yes (faster): only on access by the same OS
    blocking locks = no    # yes: depending on the used operating system
                           #  a F/RLOCK() failure will not return .F.
                           #  but may wait until record/file is unlocked
example:
[data]
    path = /usr/data/common
    read only = no
    create mask = 0666
    directory mask = 0775
    locking = yes
    strict locking = no
    oplocks = no
    blocking locks = no
```

Windows Server/Workstations: Disable oplocks (opportunistic locking) in registry when accessing data concurrently from different operating systems (e.g. Unix + Windows), see details in MS KnowledgeBase Q129202 on http://support.microsoft.com/default.aspx?scid=kb;en-us;129202

Novell Server: See Windows server above for oplocks and/or Novell TID10085899 on http://support.novell.com/cgi-bin/search/searchtid.cgi?/10085899.htm

Permission/access rights

When you need to have read and write access to databases and indices, the user needs to have at least

- read+write+execute (rwx) access rights to the used directory, and
- read+write (rw) access to databases (.dbf, .dbt, .dbv) and indices (.idx)

For read-only access to databases and indices, the user must have at least

- read+execute (r-x) access rights to the used directory, and
- read (r) access to databases (.dbf, .dbt, .dbv) and indices (.idx), and
- the database must be open in READONLY mode.

See additional details in LNG.3.3. The same apply for MS-Windows (file and/or directory property in Explorer) except there is no explicit execute permission available. Neither the directory nor files must have hidden (H) or system (S) attributes.

File and path names

On DOS, the file name consists of the name (1..8 characters), a dot "." and an optional extension (0..3 characters). Directory (path) names are separated by a backslash "\". File and path names are not case-sensitive.

All versions of Unix support at least 14 characters (usually 250 or more) in the filename, same as Windows 32/64bit versions. In Unix/Linux, lower and upper characters are distinguished, and dot (.) is treated as any other character, i.e. it has no predefined meaning and can be repeated in the filename as many times as selected. In Unix the filenames are usually given in lower case characters, directory and paths are separated by a slash "/".

FlagShip for Unix/Linux automatically translates the given backslash "\" character to "/" during a file access, FlagShip for Windows translates "/" to "\", so you can use common programming and sources, independent on the target operating system. When the lower/uppercase translation is specified using FS_SET("lower"|"upper",.T.) or #include "fspreset.fh" invoked, all specified file names (and extensions) are automatically converted to lower/uppercase during file access. Additionally, specifying FS_SET("pathlower"|"pathupper", .T.) will then automatically translate all given path or directory names to lower/uppercase.

On Unix, an embedded space or special characters are valid within a file or directory name (to check this use e.g. "`ls -lb *`"). To avoid confusion, FlagShip removes all leading and trailing blanks from the specified file name. Using FS_SET("shortname") will also remove embedded spaces or special characters from the file and path names. If you need to access file names or paths including spaces, enclose it in double quotas "...".

Source compatibility

FlagShip ensures the greatest possible source compatibility to xBASE systems on DOS, i.e. Clipper (Summer'87, 5.x, 5.2). Also, most commands and functions of dBASE (III, III+ and IV) and Fox (FoxBase, FoxPro) are supported.

The main source differences are described in LNG.1.3 and LNG.9. In the reference sections (CMD, FUN, EXT), the significant differences between FlagShip and other systems are also specified.

FlagShip for Unix/Linux and for MS-Windows is fully source and data compatible to each other, assuming the same FS release is used.

Data compatibility

All binary files from other xBASE systems on DOS (.dbf, .dbt, .fpt, .mem, .txt, .rpr, .lbl) are 1:1 compatible. Only the indices (.idx in FlagShip) must be rebuild anew for the first time. FlagShip databases and indices are cross-compatible for different platforms (Unix, Linux, MS-Windows). Using the replaceable database drivers (RDD) package also allows the use of Clipper and Fox indices and/or access to other SQL database systems such as mySQL, Postgres, Informix, Oracle etc.

Refer also to section LNG.9 and http://www.fship.com/rdds.html for more details.

1.2 Customizing your Application

The usual process to get an independent application running on Unix/Linux or Windows is:

1. Create the program sources on Unix (or MS-Windows) using a usual text or program editor -or- transfer your available sources (and data) from DOS as described in chapter FSC.7.
2. Compile the sources and test the application as described in section FSC.
3. The final executable may be named e.g. "start" or "address" etc. to distinguish it from all other files (MS-Windows executables have always .exe extensions). In Linux, you may use any extension of your choice, e.g. ".out" or ".exu" for the executable. Note the lower and uppercase sensitivity on Unix. You may also compile statically by using the -stat compiler switch and with debugger using the -d switch.
4. If the application is to be used by other users on the same system, copy the final executable from your working directory (or make symb. link) to a shared binaries directory (e.g. "`sudo cp myexe /usr/bin/.`" or "`sudo ln -s myexe /urs/local/bin/.`") and ensure that you allow the execution for the required user, group or for all users (e.g. setting the -rwxr-xr-x attributes ("`chmod 755 myexe`") for general purpose use).

   In MS-Windows, you may copy the executable to any directory of your choice, best into one included in the %PATH% environment variable.
5. Extract the re-distributable FlagShip files into your distribution directory by using the distribute.sh script or distribute.bat batch file, see next section 1.2.1.

   For distributing dynamically compiled Linux applications, you will need at least to copy the shared library from <FlagShip_dir>/lib/libFlagShip_8*.so to any directory on the target system, preferably to /usr/lib - otherwise add the lib path to the environment variable LD_LIBRARY_PATH, e.g. “export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/data:/usr/local/additional/libs”. For statically linked executable by the -stat switch, nothing else than the executable is required.

1.2.1 Re-distributable FlagShip Files

From the FlagShip Package, you may distribute with your application:

1. <FlagShip_dir>/terminfo directory & subdirectories with the terminfo entries and translation/mapping tables for GUI and Terminal i/o
2. <FlagShip_dir>/xfonts directory for GUI and Terminal i/o mode
3. <FlagShip_dir>/bin/FSwhichterm script for Unix/Linux
4. <FlagShip_dir>/bin/FSXTerm.xrdb* scripts for Unix/Linux
5. <FlagShip_dir>/bin/newfs* scripts for Unix/Linux
6. <FlagShip_dir>/bin/*tic194 optional for Unix/Linux

7. Only when your application is linked dynamically (not required when compiled statically by using the -stat switch): <FlagShip_dir>/lib/libFlagShip_8*.so for Unix/Linux, or <FlagShip_dir>\lib\FlagShip_8*.dll for MS-Windows
8. %SystemRoot%\system32\FlagShip_console.bat optional for MS-Windows

in addition to the executable of your application and required data and other utilities like start-up script/batch file described above in 1.2.4 as well as compiled tools, like <FlagShip_dir>/tools/dbu/dbu[.exe]

Other data from the <FlagShip_dir> such as the compiler, static libs etc. are **not transferable**, see also License Agreement in the <FlagShip_dir>/docu directory or in http://www.fship.com/license.html and section GEN.2

For your convenience, there is a script/batch file named **distribute.sh** or **distribute.bat** located in the <FlagShip_dir>/bin which copies the FlagShip's re-distributable files into a directory of your choice. Simply invoke e.g.

```
/usr/local/FlagShip8/bin/distribute.sh target-dir
```

or in MS-Windows

```
"c:\FlagShip8\bin\distribute.bat" target-lw-and-dir
```

or correspondingly to the installed <FlagShip_dir> path.

You may then add your executable(s) and other files into this distribution directory, tar or zip the whole directory (or create your distribution by another way like InstallShield etc).

1.2.2 Customer Settings

Your customer may copy the distributable FlagShip files (see above section) into any directory of his choice. He only need to specify the environment variable FLAGSHIP_DIR and FSTERMINFO pointing to this directory, e.g.

```
export FLAGSHIP_DIR=/home/john/flagship_distrib
export FSTERMINFO=$FLAGSHIP_DIR/terminfo
```

or in MS-Windows

```
SET FLAGSHIP_DIR=c:\my\data\flagship_distrib
SET FSTERMINFO=%FLAGSHIP_DIR%\terminfo
```

Other environment variables used at run-time are described in FSC.3.3 When compiled dynamically on Unix/Linux, you will need to set/expand the LD_LIBRARY_PATH to e.g.

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$FLAGSHIP_DIR/lib
```

or create a symbolic link (as su):

```
ln -s $FLAGSHIP_DIR/lib/*.so /usr/lib/.
```

On Unix/Linux, it is good practice to set the required terminal TERM (and other environment variables) using e.g. a shell script named, for example, "address" which then invokes the

executable named "address.out". Example of a script, invoked by "address" and located in the current or /usr/local/bin directory. It also accepts additional command-line parameters for the executable:

```
#!/bin/sh
# start script for address.out

# TERM=FSansi; export TERM    # Unix
TERM=fsansi; export TERM      # Linux
# FSTERMINFO=/usr/local/FlahShip6; export FSTERMINFO  #optional
FSOUTPUT=/tmp ; export FSOUTPUT
D_FSDRIVE=/usr/data/address/addon ; export D_FSDRIVE

if [ `expr "$PATH" : ".*/usr/bin:"` = 0 ]
then
   PATH=/usr/bin:$PATH
fi
if [ `expr "$PATH" : ".*/usr/5bin:"` = 0 ]      # for SUN-OS only
then                                            # for SUN-OS only
   PATH=/usr/5bin:$PATH                         # for SUN-OS only
fi                                              # for SUN-OS only
cd /usr/data/address                            # the applic home
echo Starting the address program
address.out $*
echo Finished. Thank you for joining.
```

The above script sets all required environment variables, checks the PATH, changes to the proper directory and executes the application. When finished, all previous settings remain unchanged. Note, also that the script has to have an executable permission (e.g. -rwxr-xr-x), set by chmod 755 address .

You also may invoke the "newfswin ..." or other newfs* scripts or create an extract from to set the environment properly, see section REL.

1.3 Porting to other systems

Owning the FlagShip developer's license, you have two options for quite simply porting the application to another operating system:

1. Purchasing another FlagShip "developer's license" for the other target system. You may directly transfer the original *.prg or the produced *.c sources to another Unix system and invoke FlagShip there to produce the executable.
2. Purchasing the economical "additional license" of FlagShip for any other system(s). You may transfer the *.prg sources, or the produced *.c sources to another Unix system and invoke FlagShip there.

All the used binary files (except *.idx, and *.o, *.a) may be transferred and re-used directly.

When the other system runs with the same processor (CPU) type, and the same Unix release, you may be able to execute the application directly on the target, without porting it. Sometimes

it may be necessary to link the application with "static" instead of using the dynamic libraries. See the compiling options in section FSC.1.3

Note: We cannot make any warranty for attempting to run an executable on a different operating system except the one where the executable was created, since Unix differs from one computer to the other and sometimes the libs are not fully compatible. Therefore, we guarantee the full functionality on the same operating system with a release number which is the same as or higher than that given on the FlagShip distribution media (see label and your Activation Card) only. See also LNG.1.3.

2. Tuning the Input/Output

2.1 GUI and Terminal i/o background

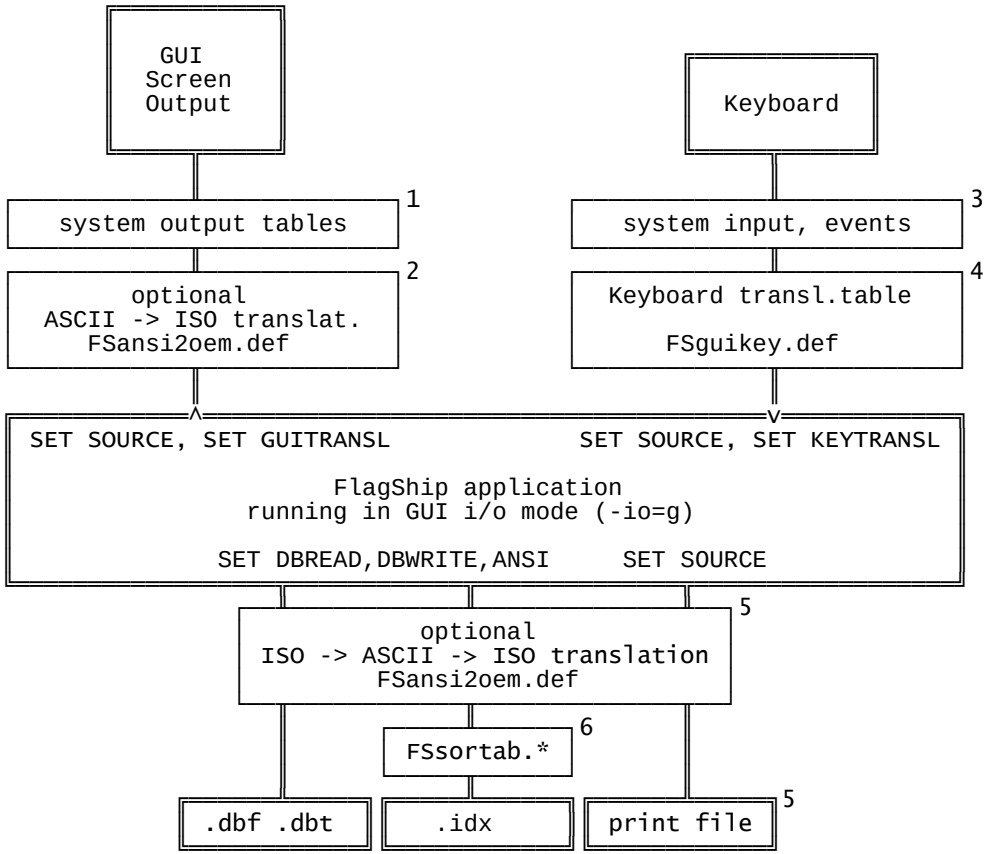
FlagShip supports three different input/output modes: GUI, Terminal and Basic i/o mode. The resulting/used mode is specified by compiler switch (-io=g, -io=t, -io=b). If none (or -io=a) was used, the application is created for hybrid mode, which means the currently used i/o is detected at run-time automatically or specified by -io=g/t/b command-line switch. Refer to section FSC for further details.

FlagShip is pre-configured for ready-to-go. The following description is meant mainly for system administrators and for developers/users who wish to understand the background or want to adapt the input or output to his special needs or to other national requirements.

2.1.1 GUI i/o mode

The GUI i/o mode apply with -io=g compiled/invoked application, or is set automatically if a hybrid-compiled application detects MS-Windows, or X11 server in Unix/Linux.

At the system level, GUI i/o behaves very differently to plain textual input/output. Any output is graphically drawn in application window or in it sub-windows named widgets or controls. The used i/o character set is ISO (or Unicode). See also section LNG.5.3 and OBJ.Application Class for further details.



Notes:

1. In Unix/Linux, the X11 system tables maps/translates input and output corresponding to settings from /etc/X11/XF86Config and ~/.Xdefaults . The output is assumed to be in ISO character set according to LANG environment variable.

   MS-Windows uses the input and output services assigned via Control Panel -> Regional and Language Setting -> Regional Options / Languages -> Details -> Text Services and Input Languages / Advanced -> Language

Note: since the PC-8 semi-graphic characters for lines and boxes are not available in ISO charset, FlagShip draws them natively when SET GUITRANSL BOX/LINES/TEXT is ON.

2. The screen output is assumed in Win/ISO character set. FlagShip will convert output chars from ASCII to ISO automatically when SET SOURCE ASCII or SET GUITRANSL ASCII ON is set, i.e. the u-umlaut character chr(129) sent by ?, ??, @.., Qout() etc. is displayed as chr(252). A manual translation is available via oem2ansi() function. You may specify your own translation table by FS_SET("ansi2oem").

3. A keyboard press sends a scan code to the system driver, which then translates it according to the keyboard layout and language used. In Unix/Linux, the translation is controlled by X11 setting and LANG environment, same as (1). On Linux, you may check and set the keyboard mapping by dumpkeys, showkey, loadkeys, keymaps or xrdb, as done in the newfs* scripts. In MS-Windows, services same as in (1) are used. A key press results in system event, cached in FlagShip by INKEY() and associated functions.

4. The system input is usually in ISO charset, i.e. chr(252) for u-umlaut, and special codes for function keys. Some systems also use Unicode. FlagShip checks any receiving key press against the FSguikey.def table and if corresponding key is declared there, translates it to expected INKEY() value. You may specify your own keyboard translation table by FS_SET("guikeys"). An additional conversion from ISO to ASCII character set is performed when SET SOURCE ASCII or SET KEYTRANSL TO ASCII is set.

5. If you prefer Windows/ISO editor but your database should remain DOS and Xbase compatible (means in ASCII format), you may specify SET ANSI ON (or SET DBREAD/ DBWRITE ANSI) to translate ISO chars to ASCII sent from the application to database (i.e. "REPLACE field with chr(252)" will store chr(129) in the database) and vice-versa for read. The same table as in (2) is used, you may specify it by FS_SET("ansi2oem").

6. You may specify your own sorting table, used also for index sort, by FS_SET("loadlang") and FS_SET("setlang"). See also chapter SYS.2.6

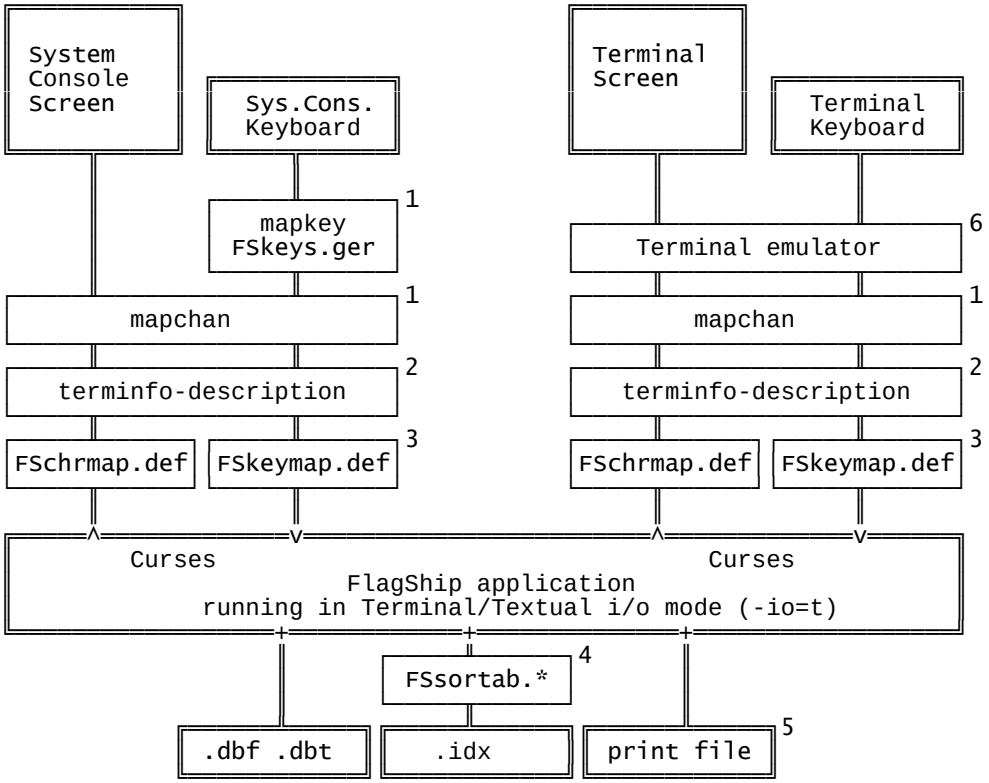
2.1.2 Terminal i/o mode

The Terminal i/o mode apply with -io=t compiled/invoked application, or is set automatically when a hybrid-compiled application does not detect X11 server in Unix/Linux.

At the system level, Terminal i/o behaves differently to GUI oriented input/output. The output on a plain console or in console-window or via remote access (ssh, telnet, terminal emulator) contains only characters or PC-8 semi-graphic characters like lines or boxes, presumed the used font support them. Since this is usually not the case for ISO-8859 font used in X11 console-windows in Unix/Linux, FlagShip provides special font images containing them. By invoking the application via "newfswin a.out -io=t" instead of "a.out -io=t" in X11 console window, these special fonts are used automatically.

In MS-Windows, you will need to select corresponding font by Properties of the Console window to be able to display semi-graphic characters and select also corresponding CodePage, see below.

In Terminal i/o mode, the input/output interface is handled by standard curses library (or ncurses, pdcurses resp.) using terminfo entries. It allows you to use or specify different kind of terminals by setting the TERM or FSTERM environment variable (see also section FSC.4). The terminal definitions are usually in the source file FStinfo.src (see SYS.2.2 and 2.3 for further details). Free modifiable input and output translation tables are available as well.



Notes:

1. Unix interface. Not available on all systems mentioned, sometimes called "ttymap". See also "Notes" for your operating system. It is sometimes recommended to disable the character mapping from mapchan (or ttymap).

   Check also the correct tty settings using "stty -a" or set them to:

   tab3 Use spaces instead of tabs, important for background color
   -istrip Do not strip the 8th bit
   cs8 Transfer as 8-bit-character if possible. Otherwise map the 8- bit characters to 7-bit within FSchrmap.def

   In Linux, the keyboard mapping table is handled by "loadkeys". The output mapping depends on the used environment character set: in graphical interface like KDE or Gnome, usually ISO or Unicode is used.

   MS-Windows does not use modifiable drivers, but uses input and output services assigned via Control Panel -> Regional and Language Setting -> Regional Options / Languages -> Details -> Text Services and Input Languages / Advanced -> Language

   Note: since the PC-8 semi-graphic characters for lines and boxes are not available in standard ISO charset used in X11 console of Unix/Linux, FlagShip uses own character images msfg*.pcf assigned via "newfswin" or manually by "xset ...", see section REL for details.

2. Unix terminfo interface to Curses. Definition of screen and keyboard sequences. Roughly comparable to "user defined ANSI driver" on DOS. See more in chapters SYS.2.2 and SYS.2.3. Refer also to (FUN) FS_SET("term") and the environment variable TERM described in the section FSC.3.3.

   In MS-Windows, the currently set CodePage of the Command Prompt Window handles the requested keyboard and output translation in Terminal i/o mode. It may be assigned by "MODE CON codepage select=nnn". Display of special and national characters depends also on used font in Window- Properties -> Font

3. FlagShip interface to Curses (FSchrmap.def and FSkeymap.def). You may redefine or remap the output and input characters, or specify the character set. See more in chapter SYS.2.3 and (FUN) FS_SET ("mapping", "outmap", "inmap").

4. Optional FlagShip interface (FSsortab.*): language dependant sorting and standard output. See chapter SYS.2.6 and FS_SET ("loadlang" and "setlang").

5. Printer output. See chapters LGN.3.4, LNG.5.1.6 and the FS_SET ("print") function.

6. Probably terminal emulator software (e.g. on PC) or hardware or other graphic/user interface (e.g. X/Open).

The environment variable TERM determines your current terminal. This variable is usually set (and exported) within the login script /$HOME/.profile (or in /$HOME/.bash or /$HOME/.login or /$HOME/.cshrc resp.) which corresponds to AUTOEXEC.BAT from MS-DOS.

You can check, set or modify this setting every time using:

```
$ env
$ printenv

$ TERM=ansi;export TERM        for Bourne and Korn-Shell, or.
% setenv TERM ansi     for C Shell
```

2.1.3 Basic i/o mode

The Basic i/o mode apply for application compiled or invoked with -io=b switch. It is designed for simple i/o use, e.g. for Web/CGI applications. The output from ?, ??, @..SAY etc. is sent to the standard stdout device, the input is read from stdin. Screen oriented output is simulated roughly in sequential manner by CR, LF and spaces.

There are intentionally no special translation tables available here, but you may manually use ansi2oem() or oem2ansi() functions to translate national characters between ASCII and ISO.

2.2 The terminfo file

This section applies for Terminal i/o only

Each available terminal is usually described in the ASCII file /usr/lib/terminfo/**terminfo.src**. After changing or modifying it as super-user, the file must be recompiled with the "**tic**" program.

FlagShip supplies an extended terminfo file named FStinfo.src, including the support of function and cursor keys etc. See chapter SYS.2.3. You may also use your default terminal terminfo description when checking the following definitions (use the system routine "infocmp default" or "untic" etc. to create an ASCII file for the current terminal TERM if the source is not available):

cols#	max. no. of screen columns (80)	
lines#	max. no. of screen rows (25)	
colors#	max. no. of colors (8)	(*)
pairs#	max. no. of color pairs (64)	(*)
it#	tabs step	(*)
op=	Sequence for standard colors	
setf=	Sequence for setting foreground color	
setb=	Sequence for setting background color	
blink=	Sequence for blinking on	
bold=	Sequence for bold (intensive) on	
rev=	Sequence for inverse on	
smul=	Sequence for underline on	(*)
rmul=	Sequence for underline off	(*)
invis=	Sequence for invisible on	(*)
clear=	Sequence for clear screen	
ed=	Sequence for clear screen from cursor	
el=	Sequence for clear from cursor to end of line	
dch1=	Sequence for clear character	
ich1=	Sequence for insert character	
cub1=	Sequence for cursor 1 character left	
cuf1=	Sequence for cursor 1 character right	
cud1=	Sequence for cursor 1 line down	
cuu1=	Sequence for cursor 1 line up	
home=	Sequence for cursor to 0,0	
cup=	Sequence for cursor home	(*)
ind=	Sequence for scroll 1 line up	
ri=	Sequence for scroll 1 line down	
smso=	Sequence for begin standard modus	(1)
rmso=	Sequence for end standard modus	(1)
enacs=	Sequence for enable altern. modus	(2)
smacs=	Sequence for alternate modus on	(2)

rmacs=	Sequence for alternate modus off	(2)
prot=	Sequence for protected modus on	(3)
acsc=	String vt100 compatibility modus	(*)
sgr0=	Sequence for clear all attributes	
sgr=	Sequence for set/reset attributes	(1,2,3 **)
ht=	Sequence for tabs in 8th step	(*)
cbt=	Sequence for back tab	(**)
bell=	Sequence for bell	
kcuu1=	Keyboard: cursor up	
kcud1=	Keyboard: cursor down	
kcub1=	Keyboard: cursor left	
kcuf1=	Keyboard: cursor right	
kpp=	Keyboard: page up	
knp=	Keyboard: page down	
khome=	Keyboard: home	
kend=	Keyboard: end	
kll	Keyboard: end, same as kend	(+)
kich1=	Keyboard: insert	
kdch1=	Keyboard: delete	
kbs=	Keyboard: backspace	
ked=	Keyboard: Ctrl + Home	(+)
kel=	Keyboard: Ctrl + End	(+)
kri=	Keyboard: Ctrl + PgUp	(+)
kind=	Keyboard: Ctrl + PgDn	(+)
khts=	Keyboard: Ctrl + Cursor left	(+)
kctab=	Keyboard: Ctrl + Cursor right	(+)
kf0=	Keyboard: F0	(*)
kf1..kf12	Keyboard: F1..F12	(*)
kf13..kf24	Keyboard: shift F1..F12	(*)
kf25..kf36	Keyboard: ctrl F1..F12	(*)
kf37..kf48	Keyboard: shift+ctrl F1..F12	(*)

Notes: The entries may be given in any order. Above the sorting is given in logical groups. See more in Unix documentation: terminfo

Legend:

(*)	optional definition.
(**)	optional definition, may cause some difficulties.
(+)	extension to standard terminfo definition
(1,2,...)	see character mapping (SYS.2.4.4.c)

Notes:

sgr this sequence has higher priority than the sequences enacs, smacs, rmacs, prot, blink, bold, rev and so on. Using the sequence mentioned above instead of "sgr" is recommended; if you do so, mark the sgr line as comment.

sgr0 this sequence has higher priority than smso

For your convenience you may also use predefined settings for the most common terminals included in FStinfo.src (see SYS.2.3 and section REL: Predefined Terminals).

To modify a terminal description, follow these steps:

1. Run the program <FlagShip_dir>/examples/fscheck.prg to find out which entry in the terminfo description requires modification.
2. Create a text file containing the definition. To do so, copy the delivered description <FlagShip_dir>/terminfo/FStinfo.src, or execute untic or infocmp to create a new file:

   ```
   $ infocmp FSVT100 >myfile
   ```

3. After editing this file, use the terminfo compiler tic as super-user to activate your modification:

   ```
   $ vi myfile
   $ su
   $ tic myfile
   ```

If changing the name of your description (i.e. when creating a new terminal description), you may wish to define or modify a character input or output mapping for that terminal (see SYS.2.4).

2.3 FlagShip terminal description (FStinfo.src)

This section applies for Terminal i/o inly

To support the highest level of compatibility between several terminals, FlagShip uses the standard Unix library **curses** and the terminal descriptions for it, which are defined in an ASCII file (usually terminfo.src) and compiled with **tic**. With FlagShip we supply a modified file called **FStinfo.src** . It is installed and compiled during installation in the directory <FlagShip_dir>/ terminfo and in /usr/lib/terminfo. It contains many extended terminal descriptions (see "Predefined Terminals" in section REL), e.g.:

FSansi, FSansi-c	for ANSI terminals 80x25, color
FSansi-m	for ANSI terminals 80x25, mono
FSansi-43	for ANSI terminals 80x43, color
FSAT386, FSAT386-M	for ANSI terminals 80x25
FSVT100	for VT100/200/300 terminals
FSvt100-m	for VT100 and VT102 terminal emulation

If you have difficulties with your standard terminal on cursor positioning, keyboard input, color output or 8-bit character set, try to assign the above terminal using e.g.:

```
$ TERM=FSansi;export TERM          for Bourne and Korn shell,
% setenv TERM FSansi               for C shell
```

You may also enter this line in your login or profile script (/$HOME/**.profile** or /$HOME/.bashrc or /$HOME/.login or /$HOME/.cshrc).

Note: Some Unix systems require different definitions than others, even if the same terminal is connected. Due to these differences, not all delivered terminal descriptions can be used on all Unix hosts without modification (see SYS.2.2).

2.4 Output mapping (FSchrmap.def)

FlagShip gives you additional features to customize your terminal output using special "character mapping". The character equivalents for input/output and the adequate character set is defined in an external ASCII file, so you or your customer may modify it, as required.

a. At the program beginning FlagShip looks for the file "FSchrmap.def" in the current directory, and if not found, in all PATH directories. If the file is present, it checks the entry for your current TERM terminal setting (such is ansi, FSansi, vt100, FSvt100 etc.). If found, it sets this mapping automatically. Note: the file FSchrmap.def is installed into <FlagShip_dir>/terminfo by default.

b. You may assign, disable or change this mapping during your program execution using the function FS_SET ("mapping", filename [,terminal]).

The file containing character mappings may have any name (see above 9.2.2.b). Example of this file:

```
# Character mapping for FlagShip                                    (1)
                                                                    (2)
:ansi                Standard PC Terminal   (3)
  0:       32  1                                                    (4)
  1-31:    *   3                                                    (5)
  32-126:  *   1
  127:     32  1                                                    (4)
  128-254: *   1
  255:     32  1
                                                                    (2)

:FSansi|FSansi-c      Standard PC Terminal  (3)
  0:       32     1
  1-26:    *|128  2                                                 (5)
  27:      60     1
  28-31:   *      3
  32-255:  *      1
:vt100|FSvt100|FSvt100-c  std. VT100 Terminal                       (3)
  3-31: ...
```

Rules and notes, see also (no) above, for the mapping description:

1. Comment lines start with the "#" character in first column.
2. Empty lines (or spaces and tabs only) are allowed.

3. Equivalent terminal to TERM environment variable. This terminal specification begins in column 1 with ":", immediately followed by the name. You may enter more synonymous names, which are separated with "|" (OR character, 124dec 7Chex). Additional, optional comments begin after at least one space or tab.

4. Character description. This line must start with at least one space or tab. It contains:
   a. The decimal equivalent of the "from" character (which should be mapped), followed by ":" (58dec, 3Ahex) and at least one space or tab.
   b. The decimal equivalent of the "to" character (output), followed by at least one space or tab. If the character is identical with the "from" character, you may enter a star ("*"). Addition (+), subtraction (-), binary OR (|) or binary AND (&) with any number (1..255) is possible.
   c. Output mode:
      1 = output using standard character set
      2 = output using first alternative character set
      3 = output using second alternative character set

5. A character description for a group of characters. Similar to a one-char description (see 4), but the description contains the character group from-to (including). Enter decimal equivalent of the start char, "-" (55dec, 2Dhex) and the decimal equivalent of the ending character.

If the current terminal is not found in your description file, or if there is no definition for the input character, standard mapping (see "ansi" above) is used.

Consider the difference between using the "?" or "??" to the direct positioning "@" output. The line-output using "?" or "??" command is internally mapped for:

```
07dec or 07hex      bell,
09dec or 09hex      tab,
10dec or 0Ahex      line feed (LF),
12dec or 0Bhex      form feed (FF),
13dec or 0Dhex      carriage return (CR)
```

so that the definitions in the mapping file apply only for "@" output for above special characters.

2.5 Input mapping (FSkeymap.def)

Similar to output mapping, the keyboard may also be mapped, e.g. for transforming an ISO character set into an ASCII PC8 set, which in FlagShip is the default to serve the program and file compatibility to DOS.

The input mapping is done in the same way as the output mapping (see SYS.2.4), using an external file **FSkeymap.def**, which is installed in <FlagShip_dir>/terminfo. To support national 7- and 8-bit keyboards as well, the following files are predefined :

FSkeymap.def	the main input mapping table
FSkeymap.ger	input mapping for German keyboards
FSkeymap.fre	input mapping for French keyboards
FSkeymap.ita	input mapping for Italian keyboards

FSkeymap.spa	input mapping for Spanish keyboards
FSkeymap.uk	input mapping for British keyboards
FSkeymap.us	input mapping for U.S. keyboards

When prompted, copy the selected file FSkeymap.* to FSkeymap.def. FlagShip searches for the current terminal TERM within the FSkeymap.def as described for the output mapping.

The table structure is very similar to FSchrmap.def, except that the third parameter is omitted. Example:

```
# FSkeymap.def ... translate ISO to ASCII
:ISO|iso|sun|FSsun

   0 - 127 : *
   128 - 159 : *+64
   160 : 255
   161 : 173
   162 : 155
   ... etc.
   191 : 168
   192-195 : *-15
   196 : 142
   ... etc.
```

whereby the first number (or area) is the keyboard character input (as decimal equivalent of it) and the second number or expression is the produced equivalence for the IBM-PC8 character set.

2.6 User specific sorting (FSsortab.xxx)

In FlagShip, you may define user- or human language specific screen messages, upper/lower translation and sorting for the commands/ functions

```
INDEX on ...
REINDEX
SORT
REPLACE ...
GET...READ
ASORT ()
LOWER ()
UPPER  ()
ISALPHA ()
ISLOWER ()
ISUPPER ()
```

a. The definitions are described in an ASCII file of your choice. During installation, the files <FlagShip_dir>/terminfo/FSsortab.ger for German and <FlagShip_dir>/terminfo/ FSsortab.7bit for 7 bit output is copied for your use.

b. When required, call the function FS_SET ("loadLanguage") within your application to read one or more language definitions, and select, enable or disable one of them with FS_SET ("setLanguage"). Please remember to use the same language definition for indexing and SEEKing.

c. The translation will be done by FlagShip fully automatically, after it has been activated.

All tables have following entries:

```
# Comment lines                                    (1)
                                                   (2)
!! ALPHABET                                        (3)
0x00 0x00 0x00 C                                   (4)
:
0x0a 0x0a 0x0a SC
:
'A' 'A' 'a' XAPGU                                  (5)
'B' 'B' 'b' XAPGU
'C' 'C' 'c' XAPGU
:

!! DAYNAMES                                        (6)
Sunday                                             (7)
Monday
:
Saturday
!! MONTHNAMES                                      (8)
January                                            (9)
February
```

:
December

!! SCOREBOARD (10)
insert (11)
out of range
invalid date

!! DIR (12)
Database Records Last change Size (13)

!! LOGICAL (14)
Yy (15)
Nn

Rules for these tables, see also (note) mark above:

1. Comment lines begin with "#" in the first column.
2. Empty lines (or spaces and tabs only) are allowed.
3. Label for the character sort table
4. Enter all (0...255) characters in the sorting order. All entries contain 4 groups, separated by space(s):
   a. character as decimal, octal, hex value or as 'char'
   b. equivalent upper case
   c. equivalent lower case
   d. type of the character (printable, graphic, etc) - see your Unix documentation: CTYPE.
5. The order of entries is relevant to the sorting order (see also FSsortab.ger, where umlauts are sorted immediately after the single letter).
6. Label for the table with names for "day-of-week".
7. Entries for this table, from "Sunday" to "Saturday", 7 lines.
8. Label for the table with names for "month-of-year"
9. Entries for this table, from "January" to "December", 12 lines.
10. Label for the table with screen headers
11. Entries for the "Insert" key, "out of range" and "invalid date" checking.
12. Label for the line with "DIR" command
13. Screen output within the "DIR" command
14. Label for allowed logical entries
15. "Yy" : keypress of "Y" or "y" is identical to "T" (true),
    "Nn" : keypress of "N" or "n" is identical to "F" (false)"

2.7 Disabling the Curses initialization

In VFS6 and later, the -io= switch specifies the used i/o interface. With -io=t or -io=a (or without -io=) switch, the curses is initialized automatically, see also source in initio.prg. When you wish to work in basic mode (e.g. for background processing or Web application, simply use the - io=b switch).

In FlagShip 4.48, the Curses/terminfo package (part of the Unix system lib) is initialized by calling the CURSESINIT() function (available in source code in the <FlagShip_dir>/system/ cursinit.prg file).

During Curses initialization, the entire screen is automatically cleared by sending the appropriate escape sequences, according to your current TERM or FSTERM environment setting (see sect. FSC.3.3 and SYS.2).

In some circumstances, e.g. if you do not need any screen oriented input/output, or if your application runs in background w/o screen and keyboard actions, you may completely or partially disable Curses by modifying the cursinit.prg file. To do so, copy it to your local directory, modify, compile according to the file header and link the cursinit.o with your application.

The CURSESINIT() function contains following function calls:

1) fgsTakeOriginalTermio

   Stores the current terminal state to reset it before leaving the application. Enabled by default.

2) fgsInitCurses

   Performs a full Curses initialization, including keyboard and screen oriented output. Enabled by default.

3) fgsIoctl2

   Performs a partial initialization of the keyboard only, instead of the full initialization. Disabled by default.

If you disable all three calls (1, 2, 3), e.g. by placing the "*" sign in front of the CALL command, no initialization and reset is performed. You may use only sequential output via ?, ??, QOUT() etc, and cannot use INKEY(), WAIT, ACCEPT, READ or other input.

If you disable (2) only, no initialization is performed. You may use only sequential output via ?, ??, QOUT() etc, and cannot use INKEY(), WAIT, ACCEPT or other input. But you may enable the input handling by un- commenting the function call (3).

Note, that on some systems (e.g. SUN OS 4.1.x) the IOCTL invocation performed in all three functions (1..3) will stop the process in background. In such a case, you should disable all three calls, if running the application in background.

See the chapter 2.9 below for the Web use and for additional www hints.

Note: it is usually better and faster, to compile the application in Basic i/o mode using the -io=b compiler switch instead of using the Terminal i/o mode with disabled curses for such purposes.

2.8 Disabling Spooler File Creation

During program start-up, FlagShip creates the default printer-spooler file named <mainmodule>.<pid> (e.g. address.1234), containing the SET PRINTER ON output, if any. This enables you to spool the printer output in a multiuser/multitasking environment. This spooler file is created in the local directory, or the one specified by the FSOUTPUT environment variable. If no printer output was written into this file, it is deleted upon program termination. See details in LNG.3.4, CMD.SET PRINTER and FUN. FS_SET("print").

On special needs, you may avoid the automatic creation of the spooler file. To do so, disable (comment out) the statement

```
CALL fgsSetSpoolOn
```

in the CURSESINIT() function, available in the <FlagShip_dir>/system/**cursinit.prg** file. Copy this file into your local directory, make the changes, recompile and link with your application, as described in LNG.2.7.

2.9 Using FlagShip for a Web Server

Similarly, to chapter 2.7 above, you may use FlagShip also as a database for creating dynamic HTML pages. The only requirement in VFS is, that you compile it in Basic i/o mode (using the -io=b switch), that's all. You preferably may use Web*() functions available in the standard library.

In FS4.48, you will need to disable the screen initialization via CursesInit() and include (somewhere in your application) the function Programinit() which invokes fsgUse4html. The CursesInit() is available also in VFS for backward compatibility, but takes no effect there.

Example:

```
** listcust.prg
  parameter cCust
  field custID, address

  if empty(cCust)
     cCust := getenv("CUSTOMERID")
     if empty(cCust)
        cCust := getenv("QUERY_STRING")
     endif
     if !empty(cCust) .and. ("=" $ cCust)
        cCust := trim(substr(cCust, rat("=",cCust)+1))
```

```
   endif
endif
nCust := val(cCust)

dbfPath = execname(.T.)
if rat("/", dbfPath) > 1
   dbfPath := substr(dbfPath,1,rat("/", dbfPath) -1)
else
   dbfPath := "./cgi-bin"
endif
set default to (dbfPath)

?? "Content-Type: text/html"
?
? "<HTML>"
? "<HEAD><TITLE>Test Web Page</TITLE></HEAD>"
? "<BODY>"
if nCust == 0
   ? "<P>*** sorry, no data without customer ID **"
else
   ? "Listing the data for customer no. " + ltrim(str(nCust))
   ok := file("custom.dbf") .and. file("custom.idx")
   if ok
      use custom index custom shared
      ok := used() .and. !neterr()
   endif
   if !ok
      ? "<P>*** error, cannot open database or index"
   else
      seek nCust
      if !found()
         ? "<P>*** sorry, no data available for this cust no."
      else
         ? "<P>"
         while !eof() .and. custID == nCust
            ? custID, address, "<BR>"
            skip
         enddo
      endif
   endif
endif
? "<P>"
? "</BODY>"
? "</HTML>"
?
quit

// disable curses, disable start-up messages if any
// these functions are called automatically by FlagShip on startup
//
FUNCTION ProgramInit()
CALL fgsUse4html          // release 4.42.0448 and newer
RETURN NIL

// Note: this is for backward compatibility to pre-FS5 versions.
// Better is to compile in Basic i/o mode using the -io=b switch.
```

```
   //
   FUNCTION CursesInit()
   RETURN NIL

** eof
```

Additional info and examples are available in the file <FlagShip_dir>/ tools/fs4web.html or on the Web server http://www.fship.com/fs4web.html. For your convenience, you may also use the WebKit package available on the fship.com server. The **CALL fsgUse4html** in the ProgramInit() UDF disables the output of additional Demo messages to stdscr; you may leave it also in the application compiled by the full licensed software.

Note: for licensing purposes, the FlagShip Demo and Personal license cannot be used for by public accessed Web pages, but for internal and test purposes only.

3. Tuning the System

The Unix operating system is built on a very modular basis. The kernel of the system may be reconfigured by the system manager (super-user or root- user) modifying the kernel variables with "configure", "idtune" or using a system administration utility (like "sysadmsh", "sysadmin" etc. if available).

Note for MS-DOS users: the Unix kernel is comparable to the resident part of COMMAND.COM and the network software on DOS. Tuning the kernel parameters is comparable to (but more powerful than) changing the CONFIG.SYS and the network configuration file, EMS/XMS memory manager, and redefining some system commands like SHARE.

Tuning the kernel is used mostly to increase the system performance, and to set or change user/system-wide restrictions (e.g. the RAM space and number of file or locks).

The file access time depends on the hardware used and the file system installed (the specific file system may be twice as fast or slower than other available file systems, especially when a high security level is used).

Since each Unix system differs slightly from all others, please refer to your system administrator documentation for details on the kernel tuning. The following hints and descriptions are given as a guideline only and may slightly differ from your currently used system. Also, the listed parameters that follow are only an extract from the tunable parameters available. The differences between SVR3 (system five release three) and SVR4 (release four) are noted where applicable.

There are no "general" kernel settings recommended, because many parameters (RAM and disk size, application requirements etc.) may differ from system to system even of the same type.

The main rule is however to ensure

- enough file tables (open files per user and system wide),
- at least one shared memory segment per FlagShip application plus whatever amount of shared memory the rest of the system requires. This should however be adequately provided by each default/shipping setting of any machine/Unix system,
- enough RAM plus swapping space, which amounts to the total memory your application will ever need for execution plus OS requirements,
- sufficiently large lock tables to accommodate the amount of users multiplied by the number of simultaneously locked databases plus 10 system requirements.

Kernel parameters

You may check the default, minimum and maximum setting values in the /etc/conf/cf.d/mtune file, and the current changes in the /etc/conf/cf.d/stune file. Warning: do not change the parameters in the file directly, but use the above described system commands instead.

Whenever you change the values of kernel parameters, you must rebuild the Unix system by the "idbuild" or "link_unix" command (or by the system administrator utility) and reboot (shutdown) the system to make the changes effective.

Before modifying the kernel, consult your system administrator manual to find out how to bring up the system in case the new kernel fails to boot. If necessary, create an emergency boot floppy or tape. Most systems leave a backup copy of the current kernel somewhere on the system. This may be used to boot using some special procedure.

Open files

The Unix operating system supports 20 to 4000 simultaneously opened files for a user (system dependent). The defaults vary between 20 and 100 and are also system dependent. The system shell uses three file slots itself (stdin, stdout, stderr).

If you have an application that uses more than the predefined number of files per user, or you frequently have I/O errors when attempting to open or create a file, pay particular attention to the parameters.

SFNOLIM Number of open files per process (SVR4). May be usually set up to 1024 (or higher).

HFNOLIM Number of open files system-wide (SVR4). May be usually set up to 1024 (or higher).

NOFILES Number of open files per process (SVR3). Note, that some older systems may ignore values beyond 60 or may crash the system in some circumstances (e.g. SCO 3.2).

NFILE Number of open files system-wide (SVR3). May be usually set up to 600 (or higher).

You may check the system tables currently used with the “sar -v” command.

In Linux, the number of currently open and max available files is handled by /proc/sys/fs/file-nr and /proc/sys/fs/file-max (view by "`cat /proc/sys/fs/file-nr`" and "`cat /proc/sys/fs/file-max`"), see details in "man 5 proc". You may increase the system-wide file limit by e.g. "`echo 100000 > /proc/ sys/fs/file-max`"

In MS-Windows, FlagShip supports at least 2000 open files per application.

Lockings

If all currently running applications exceed the predefined number of locks, you will be not able to perform RLOCK(), FLOCK(), open a database or even invoke an application. FlagShip requires up to 3 locks for each open .dbf, and 1 lock for each open .dbt, and .idx file. It is therefore recommended to ensure enough locks depending on the number of open files desired. The tunable parameter is

FLCKREC Number of locks system-wide. May be usually set to 2000 (or higher). The default is usually between 100 and 300.

You may check the currently used locks with the “sar -v 1” command. FlagShip requires up to 3 locks for each open .dbf, and 1 lock for each open .dbt, and .idx file.

In Linux, "`cat /proc/locks`" shows current locks, see also "man 5 proc".

File size

The largest file size the process may write (any file size may be read) is set by following parameters:

SFSZLIM The largest file size in bytes that the process may write (SVR4). The parameters may be set up to 2.147.483.647 (2 GBytes) or higher, up to the disk space available. The default is usually 8 MByte. Can be changed on-line by the system command ulimit (sh, ksh).

HFSZLIM The maximum of SFSZLIM, system wide (SVR4).

ULIMIT The largest file size in blocks of 512 bytes that the process may write (SVR3). May be set up to the disk space available. The default is usually set to 2.097.152 (1 GByte).

You may check the file size with the ls -l * command, the available disk space with df and the disk space used with the du command.

In Linux and MS-Windows, FlagShip supports also files larger than 2 GB (system dependant, usually up to 16 TB or even larger). For databases > 2GB use SET LARGEFILE ON which however may break backward compatibility to older databases from DOS or FS4.

Buffer caching & update

The system buffers are written (flushed) frequently to the hard disk by the bdflush daemon, according to the parameter

NAUTOUP The time in seconds for automatic system updates. The default is generally 60 seconds. Specifying a larger value increases system performance at the expense of reliability.

Flushing may be forced by FlagShip's COMMIT ALL or DBCOMITALL(). In Linux, you alternatively may invoke "sync" system command.

Processes

The Unix system processes several tasks simultaneously. Even in idle mode several tasks (daemons) are running; see “ps -afe”. You may increase the parameters:

MAXUP Number of concurrent processes per user. The default is usually 15..60; the maximal value is the NPROC setting.

NPROC Number of active processes system-wide. The default is usually 100..200; the maximal value 400 and more.

NREGION Number of regions system-wide (SVR3). Most processes have three to six regions (text, data, stack, shared memory, shared library text and data). The value should be set according to NPROC (at least times three).

You may check the current process activity by “ps -afe” and the throughput with “sar -q” command. In Linux, you also may use "top" or the /proc pseudo-filesystem (see "man 5 proc") or several GUI tools. Windows displays this by "Task Manager".

Shared memory

FlagShip uses a small piece of shared memory for inter-process communication. Each FlagShip application requires about 30 bytes of shared memory. Additional shared memory is often used by remote processes.

SHMAX Maximum size of shared memory in bytes, system-wide. The default of 524288 is generally sufficient.

SHMNINI Number of shared memory identifiers, system-wide. The default of 100 is generally sufficient.

XSDSEGS Number of shared memory segments (in XENIX only), system-wide. The default value of 25 is often reached quickly. A value of 100 is generally sufficient.

XSDSSLOTS Number of shared memory slots (in XENIX only), system-wide. The default value of 3 is generally sufficient.

Memory swapping and paging

Most current Unix systems SVR3 and up allow running a single application or the sum of all user applications which are larger than the available RAM storage. The Unix kernel swaps a part of currently inactive RAM storage to disk and loads another part of the program into RAM. The internal algorithm is effective, but beyond the scope of this manual. Usually the oldest and most infrequently used code and/or data portion is swapped out. See also the pageout or

vhand daemon description in your Unix manual (available on some SVR4 systems). Some of the swapping parameters can be changed by tuning the kernel parameters

MAXUMEM Maximum size of user's virtual memory in pages (1 KByte or 4 KByte, system dependant). May be set up to 8192..32768 (32 MB or higher). The default is generally set at 2560 (2 MB).

MAXSLICE Number of clock ticks for user process (SVR3). The default value is generally 60 (one second). On SVR4, the system command nice dynamically controls program priority.

SVMMLIM Maximum addressable size of a process (SVR4) per user. Increase the default value of 4..16 MB when running very large applications.

HVMMLIM Maximum size of a process (SVR4), system-wide.

Note: The swapping disk space is allocated during the Unix set-up process as a fixed file system and is generally almost impossible to resize later using the swap system command. Therefore, calculate carefully your application size and multitasking/multiprocessing requirements, when installing the system. A very rough rule is the maximum user number times the available (or later extended) RAM memory space.

You may check the current swapping activity with “sar -w” or “sar -q” command.

Shared applications

The Unix system loads the same application (executable) only once into the virtual memory and uses the same code block (text segment) for all users. Only the virtual data (stack and data segment) is allocated every time the application is invoked.

Calculating how many users may execute the same application is quite difficult and depends on all the other processes which run simultaneously. Therefore, the following calculation can only be used as a rule of thumb.

Steps to take:

a. Determine the size of the memory remaining when the boot process is completed. This is available as a message somewhere at start up, and also in the file /usr/adm/hwconfig. This is the memory actually available to the user and system processes after the kernel. Lets call this (**KM**) and use kilobytes as base unit.

b. With "sar -r 1 1" on an otherwise idle running system, read out "freemem", lets call it (**FM**). This is the memory in 4K blocks (SCO) still usable by user processes.

c. Now you can determine how much memory is used by the system processes. This is a place to look in when searching for more memory. Let's call this system memory (**SM**).

```
SM = KM - 4*FM
```

d. To estimate how many instances of an application can run simultaneously on a machine, you need to determine the memory usage of the application. A combination of two methods has to be used

1. issue: "size <application_name>", you will get "text + data + bss". The "text" part in bytes is the portion of the program which will be shared among the instances. "data + bss" are per user requirements. However, at runtime, additional space will be allocated.
2. To determine the current size of this space, you can use "sar" and subtract the reported freemem from (**FM**), provided that this is the only additional process running. Let's call this value application memory (**AM1**),

   ```
   AM1 = "new sar freemem" - FM
   ```

   or, use "ps -lp <pid_of_application>" and read off the value under the column SZ. This is the memory occupied by the process in 4k blocks. Let's call this value (**AM2**).

   ```
   AM2 = "value under SZ"
   ```

   Now, the per user memory (**PM**) needs actually are

   ```
   PM = 4*AM1 - text/1024   in kBytes
or PM = 4*AM2 - text/1024   in kBytes.
   ```

   These values will differ from "data + bss" and each other, and depend on what your program has done up to the point when memory is measured.

e. Now that all parameters are known, we can calculate approximately how many users (**NU**) can be fit onto our system before swapping begins, and also how many users there can be before the absolute limit is reached. (**SP**) is the swap disk size in kB.

```
NU  = (FM - text/1024)/PM          without swapping
NU1 = (FM + SP -text/1024)/PM      the limit
```

Example:

```
Total RAM memory = 16000 kB
SP = 32000 kB
KM = 13000 kB   (3000 kB used by kernel)
FM = 10000 kB
SM =  3000 kB   (used by shells and system processes)

text        = 1300 kB
data + bss  =  700 kB
AM1  = AM2  = 1000 kB --> size actually required per user
NU   = 8 users        --> very realistic, machine still
                          performing well
NU1  = 40 users       --> theoretic, but machine
                          becomes unusable
```

Note1: You must take into consideration one shell per user, so the real number of application instances will be even less.

Note2: Swapping of the data + bss sections is not so critical, since most of the text section will still have to be maintained in memory, and the data is much smaller than the application itself. A fast disk improves performance dramatically.

Note3: Disk traffic is the most critical speed factor.

Sharing code in memory is automatic (is set by default), and is controllable with "ld" options (- i -A -F on SCO); also usable with the FlagShip compiler option -Wc,...

Index

N

O

P

S

T

W

Notes

multisoft Datentechnik
Schönaustr. 7
D-84036 Landshut

http://www.fship.com
sales@multisoft.de
support@flagship.de