Short documentation for FLX

Revision 1.3, 18 September 1994

FLX is a program (written in C) that lets Unix or DOS systems read and write RSTS disks, or container files that contain disk data. It supports any size disk that RSTS supports (including "large" disks, those with device cluster size greater than 16), and any size file.

Credits

This program is dedicated to the memory of Simon S. Szeto, RSTS friend extraordinary.

Thanks to Fred Knight for providing the original inspiration (RSTSFLX for VMS); to DJ Delorie for his DOS version of GNU C; to John Wilson for Ersatz-11, PDP-11 simulator for the PC; and to Bob Supnik for his help in making FLX generally available.

Introduction

The FLX command line has the following general form:

```
command [switches... | files...]
```

The command may be given as part of invoking the program (i.e., "`flx`" followed by the command line). In that case, the command line is executed and FLX exits. If no command is given, FLX prompts for a command, executes it, and prompts for another. Switches and file name arguments may be mixed in any order; unlike many Unix commands, there is no requirement to put the switches before the file names. However, if a switch has an argument, the argument must follow immediately after the switch.

Commands and switches may be abbreviated to the shortest unambiguous abbreviation. They are case sensitive, and apart from a few switches are in lower case. There is no notion of "conflicting" switch (as DCL has) where one switch overrides a preceding switch of opposite meaning. If you include conflicting switches in a command, typically one will be ignored, but it doesn't depend on the order. So don't do that...

Most commands take one or more filespecs as arguments. These may be wildcarded. Beware of shell wildcard expansion; that general isn't what you want. So if you use wildcards, you'll usually need to quote the filenames, or to let FLX prompt you for the command line. (When FLX prompts, shell wildcard expansion is *not* done.) However, with the "put" command the input filespecs are filespecs in the host system, not on the RSTS disk, and in that case you generally *do* want to let the shell do wildcard expansion for you.

RSTS filespecs accepted by FLX generally conform to normal RSTS rules, with a few limitations and additions. The allowed field are PPN, name, extension, protection (in that order). Name and extension are *not* case sensitive.

PPN can be a normal RSTS PPN spec [*proj,prog*] or -- to avoid some shell quoting -- a Unix style spec */proj/prog/* . If no other fields follow, the trailing / may be omitted. The special PPN characters (such as $) are not supported. For most commands, the default PPN is [1,2].

Unless otherwise stated below, the file name may not be null (but the extension may be). As in recent RSTS releases, trailing ? characters in name or extension may be replaced by a *. So "`a?????`" and "`a*`" are equivalent.

The RSTS disk or container file is specified in one of three ways:

1. If the `-disk` switch is present, the argument of that switch is the disk filespec.

2. Otherwise, if environment variable `RSTSDISK` is defined, its value is the disk filespec.
3. Otherwise, container file "`rsts.dsk`" in the current directory is used.

FLX uses read-only access to the disk or container file if the command only does reading (e.g., get, list), and requires read/write access only for commands that modify the disk.

Protection code is <*prot*> as usual.

The commands are:

`allocation`	Display disk block allocation information. The first and last pack cluster and the size of each free area is displayed, as well as summary data giving amount free, amount used, and size of largest free contiguous area. If `-brief` is specified, only the summary data is shown.
`delete`	Delete RSTS files.
`dump`	Display an octal and ASCII dump (and, if -wide is specified, RAD50) of a file. If a PPN is given but the name is not, the UFD is dumped. If neither is given, you get an NFS dump of the disk. (Note that currently you cannot get a file structured dump of MFD or GFD.)
`exit`	Quit FLX (useful when it's in prompting mode).
`get`	Copy files from RSTS disk to your host system. The last argument is an output filespec. It may be a directory name, in which case you get a lot of separate output files, each consisting of the directory name plus the input filename. If `-tree` was specified, each file ends up in a host system directory underneath the one you specified, whose name is derived from the RSTS directory it came from. If the output is not a directory name, you get a single output file, containing the concatenation of all the input data. An input filespec consisting only of a PPN will cause the UFD to be copied.
`hook`	Write the boot block. First argument is the device type (default DL), case insensitive. Second argument is the file to hook (default `[0,1]init.sys`). Third argument is the file from which to read the bootstrap code (default whatever argument 2 is). All arguments are optional.
`identify`	Display summary information about the disk, such as pack label, flags, and size.
`initialize`	Initialize the disk as an RSTS file structured disk. The pack ID is the first argument; the second argument, if present, supplies the RDS level desired. Default is RDS 1.2.
`list`	Show a directory listing. Default PPN is [*,*]. Default file name and extension are *.
`mkdir`	Create a directory (non-user account). The PPN specified must not be wild.
`protect`	Change the protection code on RSTS files, or set/clear the P (no-delete) flag.
`put`	Copy files from the host system to the RSTS disk. The last argument is a RSTS filespec. Default name and extension are *. The PPN must not be wild unless `-tree` is specified. With `-tree`, the output RSTS directory is derived from the last directory name in the input filespec, which must be numeric. The input filespecs are host system filespecs.
`rename`	Rename RSTS files. The new name is given as the output filespec. A PPN is not permitted. The default name and extension are *. Multiple input specs are allowed, and all are renamed according to the output spec.
`rmdir`	Delete a directory (account, user or non-user). The directory must be empty. Wild PPN is allowed.
`rts`	Set the RTS name attribute on RSTS files. The RTS name is the last argument.
`type`	Type one or more files.

Some commands have synonyms:

`list, directory, ls`
`type, cat`
`exit, quit`
`delete, rm`
`rts, runtime`
`rename, mv, move`

Switches:

Many switches apply only to some commands, but they are recognized at all times. If you use a switch on a command where it does not apply, it has no effect.

-ascii	Force line oriented transfer mode in "get" and "put".
-binary	Force block mode transfer in "get" and "put".
-brief	Brief directory listing (name.ext only, 5 across).
-clustersize	Output file clustersize in "put", pack clustersize in "initialize". May be negative for "put", in which case pack clustersize is used if supplied value is smaller.
-contiguous	Make the file contiguous in "put".
-create	Create the container file as part of "initialize". The argument is the container size; it may be a numeric argument or the (case insensitive) strings "RL01" or "RL02".
-disk	Argument is the filename of the RSTS disk or container file. Valid with all commands.
-Debug	Print various debug information while executing the command. Valid with all commands
-end	Specifies last block number (0-based) in "dump". If the argument has a leading 0, it is octal; otherwise it is decimal. If this switch is omitted, the dump ends on the last block.
-filesize	Meaningfule only if -contiguous is specified; forces the output file in "put" to be that size. (If omitted and -contiguous is specified, the file is pre-extended to the same size as the input file.)
-full	Full directory listing (includes file attributes) for "list".
-image	Same as -binary.
-long	Same as -full.
-oattributes	Display RMS attributes in octal in "list".
-protect	Set P flag in "protect" command.
-replace	Let "rename" supersede an existing file, if the new name already exists.
-size	Same as -filesize
-start	Specifies first block number (0-base) in "dump". If the argument has a leading 0, it is octal; otherwise it is decimal. If this switch is omitted, the dump starts at block 0.
-tree	Copy directory trees (see below).
-unprotect	Clear P flag in "protect".
-wide	Do wide dump (include RAD50) in "dump".
-Write	Override normal write protection: allows writing on disk initialized as read-only, and allows deleting and modifying files that are write-protected (e.g., protection code <62>). Does *not* override the protection provided by the P (no-delete) flag.
-1column	Brief directory listing (name.ext only), one file name per line.

Some examples:

```
get -v [0,1]*.rts [1,2]*.sav .
```

Copy all .RTS files from [0,1] and all .SAV files from [1,2], and put them in the current host system directory. The resulting file names match those of the input files.

```
put /usr/bin/ls [1,2]*.sav
```

Copy "ls" to [1,2]LS.SAV. Since the extension is .SAV, the RTS name is set to RT11, and the runnable bit set in the protection code. (This is obviously a rather silly example...)

```
rename [0,1]*.tmp [1,2]*.sav *.foo
```

Rename all `.TMP` files in [0,1] and all `.SAV` files in [1,2], changing their extension to `.FOO`. (Note that, while no PPN may be specified on the "new" filename, the inputs may be in several different directories, or indeed have wildcard directory specs.)

More details on certain commands:

Get:

The "get" command can either concatenate files, or produce multiple output files. It concatenates if a single filespec is given as the output spec, and produces separate outputs if a directory spec is used.

The file transfer mode depends on whether concatenation is being done, on switches, and on the RMS attributes of the input files.

If concatenation is done, line mode transfer is done unless `-binary` is specified. (Note that the case of a single input file spec, without wildcards, to a single output filespec, is not considered concatenation. However, if any wildcards are present, that is concatenation even if only one file ends up matching the wildcard spec.)

If individual transfer is done, the transfer can be forced to line mode with `-ascii`, or to binary (block) mode with `-binary`. Otherwise, the properties of the input file control how that particular file is transferred:

- No RMS attributes (RSTS native format):
  - ◊ Text file -- line mode
  - ◊ Others -- block mode
- Copying a directory - block mode
- RMS sequential format:
  - ◊ Fixed 512 byte records -- block mode
  - ◊ Otherwise line mode
- Other RMS organizations -- block mode

(Note that line mode transfer other than for Stream format files is currently unimplemented. More precisely, line mode transfer assumes that the file is a stream format file, even if RMS attributes are present that say otherwise. However, it will work correctly for `stream_lf` format files, i.e., files with only line feed as line terminator.)

A "text file" is one whose extension indicates it contains text, as opposed to binary data. The extensions that are considered "text" are:

```
txt lst map sid log lis rno doc mem bas b2s mac for ftn fth cbl
dbl com cmd bat tec ctl odl ps c h
```

In line mode, the input file is interpreted as a RSTS stream file, with cr/lf line delimiters. Lone line feeds without preceding carriage return are also treated as delimiters. In any case, the lines are written to the output file using standard Unix line delimiters ("newline"). Any trailing nulls in the file are ignored.

In block mode, the input file is read block by block, and is copied exactly as is to the output file in binary mode. In this case, the output file size will be a multiple of 512 bytes.

Tree transfer mode:

If `-tree` is specified, the input directory name is made part of the output spec (this is meaningful only when doing individual transfer and the output spec is a directory spec). The PPN is converted to a 6-digit string and that directory name and the filename are appended to the output spec. If necessary, the directory is created. So the command

```
get -tree [1,2]*.sav .
```

would create (for example) a file `./001002/pip.sav` .

Put:

The "put" command does transfer mode selection similar to "get", but since there are no attributes on the input files, the rules are simpler:

The transfer can be forced to line mode with `-ascii`, or to binary (block) mode with `-binary`. Otherwise, the properties of the input file control how that particular file is transferred:

- Text file -- line mode
- Others -- block mode

In line mode, the input file is read line by line. At each line ending, a standard RSTS line delimiter (cr/lf) is inserted. Normally, the input file has "newline" characters (line feed) at end of line, per Unix convention. If the input file has cr/lf pairs at end of line -- for example, if it was a RSTS text file copied in block mode from somewhere -- then the cr/lf is kept as the line end; no additional cr is inserted in that case.

In block mode, the file is read in binary mode, and is written exactly as is to the output RSTS file. If the input file size is not a multiple of 512 bytes, the last block is filled with zero bytes (nulls) before being written.

The output file protection can be specified explicitly. If it isn't, it defaults to 60 for non-executable files, 124 for executable files. Executable files are recognized by their extension (checked against the runnable extensions of the runtime systems I know of). An executable file will always have its RTS name set. The executable bit in the protection is turned on only if the protection was defaulted.

Large files (those of size greater than 65535) have the RTS name cleared, and the executable bit in the protection code is forced off unconditionally.

Tree transfer mode:

If `-tree` is specified, the output PPN is normally wild. The input file spec is then used to construct not just the output file name but the output PPN as well. The last directory name in the path is expected to be a numeric string. If the output spec is of the form [*x*,*] then the numeric (decimal) value of the directory name is taken as the programmer number. If the output spec is [*,*x*] or [*,*] then the directory name must be at least 4 digits; the low order 3 digits are the programmer number and the high order digits the project number. (In the case of [*,*x*] the programmer number is then ignored.) Examples:

`put -tree 10/foo [2,*]`	produces `[2,10]foo.`
`put -tree 10/foo [*,*]`	is an error (too few digits)
`put -tree 4010/foo [*,*]`	produces `[4,10]foo.`
`put -tree 4010/foo [*.2]`	produces `[4,2]foo.`
`put -tree bar/foo [2,*]`	is an error (directory not numeric)

If necessary, the output directory is created as part of the copy.

What's on the kit

The FLX kit contains:

- Source (`.C`) and header (`.H`) files for FLX.
- A `Makefile`, to build FLX under Unix or DOS (with GNU GCC, or probably with any other reasonable C compiler). This file may need minor changes; read the notes in the file for details.
- `FLX.EXE`, a DOS executable file. This file was built with DJ Delorie's GCC package, but it does not depend on having that installed.

- `FLX.DOC`, the source file for this document (Microsoft Word format), and `FLX.PS` and `FLX.TXT`, readable copies of same.

Futures

Several features are on the list for consideration in the future. Feedback would be appreciated on these or other candidates (but no promise is implied that any of this will be done, or that there will be a future release).

Commands:

`rebuild`	"Rebuild" (a.k.a., "clean"), comparable to the ONLCLN utility or Refresh Rebuild feature in RSTS INIT.

Switches:

`-query`	Ask about transferring each file in commands that do file operations.
`-r`	Do RMS variable length record format transfer in "put".
`-rv`	Same as `-r`.
`-rs`	Do RMS stream record format transfer in "put".
`-rc`	Do RMS stream-CR record format transfer in "put".
`-rl`	Do RMS stream-LF record format transfer in "put".