

## OVERVIEW

The *TPF cross-development environment* is a set of OS/390 UNIX System Services (hereafter, *OS/390 UNIX*) tools that support the development of TPF e-type segments and load modules (DLM, DLL, and LLM).

This environment is implemented on a mixture of native OS/390 UNIX and custom-written commands to support all necessary steps on the OS/390 UNIX *hierarchical file system* (HFS), with support added in necessary places for traditional MVS cataloged data sets, such as *partitioned data sets* (PDS).

This page is an introduction to the cross-development environment as of December 2001 and explains its design and implementation philosophy.

## BACKGROUND

TPF, like most IBM mainframe operating systems, has historically been delivered in non-stream file systems, exploiting primarily the OS/390 partitioned data set. Many of the TPF support utilities have been written to exploit OS/390 *access methods* such as BPAM, BDAM, and QSAM, which present data in a *record* or *block* orientation. This manner of performing I/O is in diametric opposition to the *stream* implementation of an OS/390 HFS, which presents all files as a stream of bytes with no logical or physical record divisions supported by the operating system.

TPF development practices are forced to change with the times --- to a UNIX-, stream-based environment --- exactly that supported by OS/390 UNIX. The beauty of the OS/390 UNIX implementation is that it retains use of the EBCDIC character set, and it is indeed possible (with some work) to access the traditional MVS cataloged file system from the OS/390 UNIX shell. Another strength of the OS/390 UNIX platform is that it enables some terribly powerful development tools, among them *make*, which can build entire systems with a single shell command.

The ubiquitous *make* command is designed to work with *command line utilities* for common purposes such as compilation and link-editing. However, TPF development has always possessed some rather unique steps, and the entire rationale for the existence of this cross-development environment is to make it fit comfortably with *make*. Therefore, the custom tools written for TPF development are all command line-based, and return nonzero return if an error occurs -- integral behaviors that *make* requires to work properly and exhibit correct behavior with respect to its own execution.

## DUELING FILESYSTEMS

The TPF development environment is, as of this writing, in a transitional state from a JCL and MVS file system-based environment to one based on *make* and stream files. Some of the utilities still depend on PDSes. Unfortunately, *make* cannot make age-based judgments against any traditional MVS cataloged files because time stamps either do not exist, are not consistent, or are not granular enough to use. Therefore, *it is preferable to keep as many of the files produced by the development process on the HFS file system for as long as possible.*

As a result, the design philosophy of the TPF cross-development tools has been to handle HFS file residence as a default or *preferred* condition, with access to PDS members provided whenever necessary or convenient.

**It is a stated goal, for TPF development purposes, to eventually make the PDS obsolete.** In the meantime, it is seen as a necessary evil to permit PDS access. Users of the TPF cross-development environment are warned that vestiges of PDS support will slowly disappear over time. Users are similarly advised to get used to dealing with HFS files as soon as possible. One of the design tactics employed is to make PDS access, wherever necessary, as *transparent* as possible, although it is not yet possible to obsolete it entirely.

## MAN PAGE HOW-TO

The *man* (refer to *man(1)*) command displays manual pages, which contain all sorts of useful information about system commands, including those delivered with the TPF cross-development environment.

The *man* command uses what is called a *pager* to keep long pages from scrolling off your screen. The default pager is a command called *more(1)*, which responds to some basic keystrokes (you don't have to press the *Enter* key unless you are running under the OMVS 3270 shell from TSO); here is a short list of the important ones:

- q** quits the current session, returning you to the shell command prompt.
- d** Scrolls a half-screenful *down*.
- f** Scrolls a full-screenful *forward* (down).
- u** Scrolls a half-screenful *up*.
- b** Scrolls a full-screenful *backward* (up).
- g** Moves to the first line of the man page.
- G** Moves to the last line of the man page.

Refer to the *more* man page for other things you can do with the 'more' command. *more* also accepts an 'h' keystroke, which opens a help display.

You will often see a number in parentheses following a man page reference, because man pages are organized in *sections*, and the number in parentheses refers to the section in which the reference resides. The manual pages for the TPF cross-development environment follow this general paradigm:

- 1** User shell commands.
- 2** System calls.
- 3** Subroutines.
- 5** File formats and layouts.
- 7** Miscellaneous information that defies categorization; like this page.
- 8** Administrator information.

The only reason you need to know about these sections is that sometimes man pages in different sections might have the same name, and we will need to resolve the seeming ambiguity. In such an event, the section number can be passed to the *man* command as shown in the following example, which will locate the man page for *link* in section 1:

```
man 1 link
```

In this way, we can force *man* to give us information about the *link* user shell command, not the *link* function call found in Section 3.

## ASSEMBLER HOW-TO

The following instructions apply equally to TPF e-type assembler segments and general assembler usage for load module inclusion.

The output from the TPF-unique assembler command `-- as --` is the *object code file*, which always has a *file suffix* of ".o". If the *as -o* option is not used, the file name of the object code file will be the same as that of the source code file, except that it will have a ".o" file suffix and it will be written to the current working directory.

Until the *object code file* is made into a loadable image, the only difference between a TPF e-type segment

and any other type of TPF assembler program is the macros that it calls and the register mapping conventions that it respects. The same assembler (refer to *as(1)*) is used to assemble both.

Other file suffixes that the *as* command might generate are:

**.lst** This is the assembler listing of the program.

**.ADATA**

This is the debugging image of the program that is generated only if the *as -g* option is used.

The next step you take depends on whether you are building an e-type segment or a load module.

If you are building a load module, the object code file generated by *as* will be the primary input to the linkage editor (refer to *ld(1)*). If you are building a TPF e-type segment, the object code file needs to be passed through the *TPF offline loader* (refer to *tpfldr(1)*). Under **no** circumstances should a standard e-type segment ever be processed by *ld*.

## C/C++ HOW-TO

Use the *c89(1)* command (and its relatives: *cxx*, *cc*, and *c++*) as you normally would. Refer to *c89(1)* or the standard IBM OS/390 UNIX documentation for details of its usage. The TPF cross-development environment software provides no special support for this function.

C/C++ programs are never considered e-type segments. The object code files produced by the C/C++ compiler *must* be processed with the linkage editor, *ld*.

## TPFLDR HOW-TO

The *tpfldr(1)* command invokes the TPF offline loader. It is used to generate *loadsets*, which can be processed by the TPF *ZOLDR* or *ZTLDR* prime CRAS commands.

The implementation of *tpfldr* under the OS/390 UNIX shell is merely an invocation of the JES batch TPFLDR40 process, having done some dynamic DDname invocations for you before launching it. What is different about the OS/390 UNIX shell version of *tpfldr* is twofold:

- o Environment variables are used as an option to point *tpfldr* to HFS-resident e-type object files and load modules that are to participate in the loadset (among other things)
- o TPFLDR40 SYSIN *must* reside on HFS.

*tpfldr* is capable of writing the loadset either to a TPF general data set (GDS) or punching it to a VM reader. GDS usage is preferred, because there are certain installation-dependent idiosyncracies that can be introduced when using the MVS puncher to write to RSCS. In particular, the punched spool file class will probably be incorrect for *ZOLDR* or *ZTLDR* processing and may require VM CP operator intervention to change the class before it can be loaded successfully.

## LINKAGE EDITING HOW-TO

Link-editing creates TPF load modules, of which there are three different types:

**DLM** Dynamic Load Module. This is the TPF counterpart of a normal program load module. DLMs have only one *entry point*.

**DLL** Dynamic Load Library. This is a collection of routines, each having its own external *entry point*. At the time the first routine in a DLL is called, the external linkages to the remaining routines in it are also resolved, making it available to all processes on the system. When a DLL is first created by the linkage editor, it creates a *definition side deck* (or DSD), which contains a line for each routine in the DLL. Programs referring to DLL-resident routines need to include the DSD as linkage editor input so that the TPF run-time loader can have an idea of which DLL to load.

**LLM** Library Load Module. This, similar to the DLL, is a collection of functions, but each function does *not* have its own entry point. The concept of the LLM predates the DLL, and the linkage is

quite different; each entry has an *ordinal number* assigned to it by *libi(1)*. Calling programs link to *stub routines* that know the ordinal number assigned by *libi*. See "LLM HOW-TO" (below) and *libi(1)* for more information. It should be noted that the LLM is *deprecated* -- developers should not package new functions in LLMs. Use the DLL instead for this purpose.

The *ld(1)* command builds all three types of load modules. It is another TPF cross development-environment-custom written command that calls the OS/390 C/C++ prelinker and *Program Management Binder* (previously known as the linkage editor), performing dynamic DD allocations for you under the covers based on the content of the command line, shell environment variables, or both.

## LIBI HOW-TO

For those who must introduce new functions into an LLM, *libi* remains an indispensable tool. *libi* accepts as its sole input a *library script*, and from that generates a *transfer vector* and a set of *stubs*. *libi* works by building a table of branch addresses based on the *function number* and name of each function defined with the library script *libfun* directive, and then builds a *stub* for each, which passes the function number into the transfer vector, which becomes the main entry point for all functions in the LLM. See the man page for *libi-script(5)* for details on build script format requirements.

The *libi* process is unnecessary for DLMs, DLLs, and type E-segments. Those who must use *libi* are referred to the man page for *libi(1)*.

The *transfer vector* is a generated object code file that redirects execution from a single LLM entry point to the correct function contained in the LLM. The transfer vector object file generated by *libi* should be statically linked with *ld(1)* into the final LLM, and should appear as the first object file; *ld* handles the correct final ordering as long as the transfer vector object file appears first on the *ld* command line.

*libi* also generates a set of *stub object files*, one for each function in the LLM. These object files have a file suffix of ".o" and are merely named with the function name, presented in lowercase. *libi* writes each stub object to a subdirectory to correctly isolate them from the LLM components. So that modules calling this LLM can also be linked with the *libi*-generated stubs, it is recommended that the developer package them into a UNIX *archive file* with the standard *ar(1)* command, and then refer to that archive when linking referring programs with *ld*'s **-l** and **-L** options.

*libi* provides no option to create its outputs directly to the MVS cataloged file system. Persons interested in doing so are referred to the section titled "MIGRATION ISSUES".

## LOAD MODULE RESIDENCE

Once the developer has completed development operations against the load module, a choice exists as to where the final load module will reside; as a PDS member or an HFS file.

The strong *make* orientation of these tools would imply that the final load module should reside on HFS, so that the *make* utility can see it and know its time and date of last build. This is true, and should be the paradigm that the developer aims for. As of December 2001, it is possible to use *tpfldr(1)* directly against HFS-resident load modules.

However, there are often cases that vary installation by installation where you, as a developer, must migrate the load module to a PDS member.

Unfortunately, the formats of an OS/390 load module differ depending on which kind of file system they reside; we simply cannot do a byte-by-byte copy of the HFS load module into a PDS. Instead, we must invoke the Program Management Binder to *relink* the HFS source load module into the PDS target load module. Invoking the Binder to perform this process is not a simple task, and has subtle differences depending on whether the HFS source load module is a DLM, DLL, or LLM; the entry point of the module *must* be explicitly identified to the Binder.

To satisfy this requirement, a custom command called *lmc(1)* (Load Module CoPy) was implemented. *lmc* users are strongly advised to read its man page because there are dependencies on environment variables required for its correct operation.

## MIGRATION ISSUES

If we follow recommendations, we are building components a piece at a time under HFS, where their characteristics are visible to *make*.

Eventually, some parts (for example, load modules) may need to become PDS members; and depending on your location's configuration, other files produced *should* reside as PDS members.

OS/390 UNIX support includes the **OGET** and **OPUT** commands, which move files *from HFS to the MVS cataloged file system* (OGET) and conversely, from the MVS cataloged file system to HFS (OPUT) [these commands were obviously named from a MVS point of view]. You can access these commands from the shell by prefixing them with the string "tso". For example:

```
tso oget '/u/myuid/something.c' //myuid.source.c(somethin)' TEXT"
```

moves HFS file *something.c* to its truncated PDS member name equivalent in an MVS cataloged file system. If you choose, you can move entire directory contents all at once with the "tso OGETX" command.

These move actions can easily be coded into *makefile rules*, so that they are performed only when necessary.

Do not move files from HFS to the MVS cataloged filesystem as part of the normal, everyday development cycle (that is, do not move things until you know you are done modifying or testing them) -- the *make* utility has no knowledge of age/existence attributes of any MVS cataloged file system object. If you depend on such parts, you will want them on HFS where they can be seen -- otherwise, you'll be needlessly building things because *make* will think a file named as a dependency does not exist.

## MAKING *make* WORK FOR YOU

The entire purpose of the TPF cross-development environment is to enable a *make*-based development paradigm. You can completely automate your development tasks, including the TPF-unique steps, using *make*.

*make* is a rule-based program that has many intricacies and we will not get into its details here.

There are also many versions of *make*. The TPF cross-development environment has not been biased toward any particular version of *make*, however *GNU make* is strongly recommended for developer use because of its simplicity, power, and global acceptance. Ports of *GNU make* to the OS/390 UNIX platform are available from many sources, including an IBM Redbook, IBM web sites, and the Mortice Kern Systems' web site -- any of these will suffice. The particularly stubborn may want to use the *make* that comes with OS/390 UNIX -- the TPF cross-development environment does not preclude its use.

A couple of *makefile builder* utilities are also available from Open Source repositories. The most common are *GNU autoconf* and the X Window *imake* packages. While *autoconf* is indeed useful and can be adapted to most any system configuration, it requires a lot of coding into the initial *makefile* template, and may not be the ideal tool for TPF developers in your installation. Instead, the X Window *imake* utility is strongly recommended, because the initial knowledge required to create a *makefile* is much less than that required by *autoconf*. The *imake* utility is packaged in the *itools-R6.3* distribution, and usable OS/390 UNIX ports of both *imake* and *autoconf* can be found on the CD-ROM accompanying the IBM Redbook titled "*Open Source Software for OS/390 UNIX*" (MacIssac, Barany, Schandl, Tison, IBM Corporation, 10/2000, document SG24-5944-00). Potential users of both of these packages are warned that some degree of customization will be required to support TPF program builds.

## **WHO TO BLAME**

Jim Tison, jtison@us.ibm.com

## NAME

as – Assemble a program using HLASM

## SYNOPSIS

```
as [ -I [asaltrc] ] [ -g ] [ -e ] [ -h ] [ -w ] [ -k ]  
[ -o HFS_objectfile_path ] [ -O[PDS_name] ]  
[ -r[fi_lenam] ] [ -Wa,HLASM_option[...HLASM_option ]  
[ -DSYSPARM_symbol ] [ -Imacro_spec ]  
[ --listfn=fi_lenam ] [ --asmaopts=fi_lenam ]  
fi_lenam [ ..fi_lenam ]
```

## DESCRIPTION

as invokes the *IBM S/390 High Level Assembler* from the OS/390 UNIX command line.

## OPTIONS

**-I** [asaltrc]

This option controls the **listing** behavior of **as**. The default, when this option is omitted, is to produce a listing file in the current directory with the **.lst** suffix -- the default listing option is **asa**, which causes ASA control characters to be present in column 1 of the output file. You may specify **-ltrc** to force *machine control characters* to be used. If neither *asa* nor *trc* is specified, but the **-I** switch is present, no listing will be generated.

**-g** This option forces a *debugger information file* to be created for use with the *IBM VisualAge TPF Assembler Debugger*. If environment variable TPFLDR\_ADATA is not defined or set to the empty string, the resulting debugger information file will be written to an HFS file in the current working directory with the **.ADATA** suffix.

**-e** Create an EVT file in the current working directory for *IBM VisualAge TPF* use, if necessary.

**-h** Causes **as** to display its syntax to stdout. No more processing will take place.

**-w** Suppress all warnings, and handle any RC <= 4 from HLASM as if RC=0 had been received for purposes of control flow and event file generation.

**-k** Generate and keep the SYSADATA file as an MVS cataloged data set named *userid.progname.ADATA*, where *userid* is your current TSO user ID and *progname* is the stem (no path, no suffix) of the source file being assembled. Default behavior is to not generate a SYSADATA file at all unless the **-e** or **-g** option is specified. Without the **-k** option, either **-g** and/or **-e** will cause a **temporary** SYSADATA file to be generated, which will be scratched at the end of the assembly.

**-o**HFS\_objectfile\_path

Force the object code file to be written to *HFS\_objectfile\_path*. The file name represented by this option will be created if it does not exist, or overwritten if it does exist. See **OBJECT FILE OUTPUTS** for an in-depth discussion of this option.

**-O**PDS\_name

Force the object code file to be written as a member of a PDS. The operand to this option is the fully-qualified DSN of a PDS, which must already exist. See **OBJECT FILE OUTPUTS**, below, for an in-depth discussion of this option.

**-r**[fi\_lenam]

This option causes a *macro library composition report* to be written to stdout. If a *fi\_lenam* is given, it will be written to the indicated file.

**-Wa,HLASM\_option**

This option causes normal HLASM options to be passed to HLASM. More than one option token may be passed, in which case subsequent tokens must be delimited by commas with no intervening whitespace. Any options passed to HLASM as a result of this option occur *last*, and may override other HLASM options synthesized by other **as** command line options. HLASM options presented on the command line via this option take rightmost precedence over any that may be found in the **AS\_PARMS** environment variable (see **ENVIRONMENT VARIABLES**).

HLASM options DECK, LIST, ADATA, and/or TERM (or variations thereon) may not appear as arguments to this option. Control these HLASM behaviors with the -o/-O, -l, -k/-g, or -w switches, respectively.

**-DSYSPARM\_***symbol*

This option causes *SYSPARM\_symbol* to be passed to HLASM as a SYSPARM parameter. This option may be used only once per command line.

**-Imacro\_***spec*

This option causes *macro\_spec* to be included in the HLASM SYSLIB concatenation. See **MACROS AND MACRO LIBRARIES**.

**--listfn=***fi lename*

This option causes the assembler listing to be written to *fi lename* instead of its default location (*.jpgmname.lst*).

**--asmaopts=***fi lename*

This option causes *fi lename* to be passed to HLASM as DD name ASMOPTS, which permits assembler control options to be passed in an HFS file. Such options override default installation options and those passed on the command line.

## INPUTS

The *fi lename* command line token may indicate an HFS-resident file name with either an absolute or relative path. Such files may have any file name suffix, and must contain S/390 HLASM source code. More than one source code file may be named, in which case users should avoid the **-o** option.

## OBJECT FILE OUTPUTS

Because TPF development is in a transitional state and HFS files are not fully supported, it is sometimes necessary to write object file output as a member of a PDS; this is inevitably true for TPF E-type segments that must be processed by **tpfldr(1)**. The **-O** ("big OH") option takes as its operand the fully qualified DSN of a PDS *without* member specification. When **-O** is specified, a member will be written to the indicated PDS which is identical to the *stem name* of the input source code file, folded to uppercase. Users invoking this option are warned that limitations exist on what is considered a valid member name, and that **as** will make no attempt to prevent you from using the **-O** option with any input source code file, regardless of its stem name value.

The PDS DSN argument to **-O** is optional. If none is present, the **TPFLDR\_OBJLIB** environment variable will be interrogated, and the first (leftmost) data set used in lieu of the PDS DSN argument, if present.

The default behavior, if neither the **-O** nor **-o** options are present on the command line, is to write a file with a stem name equal to that of the source file's stem name and suffixed with **.o** to the current working (HFS) directory.

The **-o** ("little OH") option can be used to cause **as** to write the object code file to some other HFS location. When **-o** is invoked, the absolute or relative path name of the object file *must* be provided as an argument.

When **GOFF** is passed to the Assembler as a runtime option, **as** reacts to it. Users of the **-O** option are warned that the output format of the object file will depend on the DCB characteristics of the DSN targeted. Users of the **-o** option are advised that the output will be created in RECFM=VB DCB mode: there will be no GOFF continuation PTVs as a result, as variable record format means that the Assembler will not need to 'chunk' up the output into 80 byte records. It is not possible to force **as** to operate in any other manner with GOFF output to HFS.

GOFF output also requires the LIST(133) option; when it is not present, HLASM returns RC=2 and forces LIST(133). **as** is sensitive to this condition and forces LIST(133) so that the user has a chance of receiving RC=0 on completion.

## MACROS AND MACRO LIBRARIES

HLASM itself is not capable of accepting macro libraries (DD name SYSLIB) from anything other than a PDS. However, working in the OS/390 UNIX environment demands that HFS macros be supported. We also need control over which PDSes are used as macro libraries and their concatenation order, which determines the order of precedence in the event of name collision.

**Support of HFS macros** is done by creating one or more *temporary PDS* data sets and populating it/them with copies of HFS macros as specified by the **-I** option, when the argument names a HFS directory or explicit HFS file name. When an HFS *directory* is named as the operand to the **-I** option, all files in that directory are scanned, and those with suffixes of **.mac** or **.cpy** are added to the temporary PDS. When an HFS *file name* is given as the argument to **-I**, that file name must end with the **.mac** or **.cpy** suffix, and it alone will be added to the temporary PDS.

The temporary PDS is cumulative, and all member names in it must be unique. By "cumulative", we mean that a temporary PDS is created on the first occurrence of an HFS **-I** specification, and continues to amass copies of additional HFS specifications until either a PDS DSN is named with the **-I** option, or no more macro specifications are found on the command line. Member names are derived from the HFS file name of the eligible macro by stripping any leading path information and suffix information or both, then folding that value to uppercase and swapping "\_" to "@" characters. If the derived member name is not a legal PDS member name, a message stating this will appear on stderr, and the macro will not be copied into the temporary PDS. Otherwise, the macro will be copied. Users should note that this can be an **I/O-intensive operation**, and should take place only for a minimal number of HFS-resident macro files. **as** will allow no more than 1,300 members to be inserted in any temporary PDS. *Uniqueness* is controlled internally by **as**, by disallowing membership of any second (or subsequent) identical member name in a *single temporary PDS*. When such *collisions* occur, a message will be sent to stderr describing it.

**Control over the macro library concatenation order** is done by two means: the **-I** command line options, handled in left-to-right order (leftmost is highest precedence); and handling the command-line specified macro libraries with higher precedence than those named in environment variable **AS\_SYSLIB**. **AS\_SYSLIB** is handled as a colon-delimited list of PDS DSNs that are to participate in the final SYSLIB concatenation, with PDSs appearing leftmost (first) in the list having higher priority than those toward the right. Users may set **AS\_SYSLIB** to a list of "base" macro libraries that they commonly use, or have **AS\_SYSLIB** exported via a *makefile*, and then only extend this "base" set with exceptions from the command line.

**The DCB attributes of temporary PDSs** are optimized for 3390 DASD and by are by default assigned to UNIT=VIO for performance reasons. Specifically, the following attributes are used:

- o LRECL=80
- o RECFM=FB
- o BLKSIZE=27920 (3390-optimized)
- o DSORG=PO

**Space allocation for temporary PDSs** is based on the anticipated number of tracks that the actual data is likely to occupy based on the sum of actual HFS file sizes to be included as PDS members plus a 50% record padding factor; this padding factor assumes that the stream nature of HFS text files will cause approximately two-thirds, on average, of the 80-byte record to contain actual data. We make this rather drastic assumption because we will use this temporary PDS in a data set concatenation, and it is important that all data fit in the first extent; overflow into secondary extents limits the number of PDSs that may participate in the concatenation. Every attempt is made to have all HFS file images in the PDS occupy only the primary extent; however, secondary extents are allocated and sized using a formula of  $((.05 * \text{datasize}) + 1)$  tracks. (Most commonly, this works out to a secondary extent value of 1 track.)

## SYSADATA

Some users find it necessary to generate SYSADATA with the **-k** switch, which causes an MVS cataloged data set to be created that contains the data produced by HLASM. By default, space is allocated for SYSADATA data sets with a primary extent of 3 cylinders and a secondary extent of 3 cylinders. This allows for a

SYSADATA data set, by default, to occupy a maximum of 48 cylinders, with any unused space FREEd. By their very nature, certain unusually large assemblies, such as TPF Control Program CSECTs, overflow this default limit: applications programmers will rarely run into this problem. In order to resolve this problem (and the ABEND B37 that is its symptom), **as** has two optional switches, **--sysadataprisp=**, and **--sysadatasecsp=**, whose operands are expressed as a number of cylinders to allocate for the primary and secondary extents for the SYSADATA data set, respectively. For example, if **--sysadataprisp=15** and **--sysadatasecsp=15**, as many as 240 cylinders would be allowed for the SYSADATA data set in the presence of the **-k** switch.

When **-k** is present, the SYSADATA data set is kept and cataloged under the data set name *userid.progname.ADATA*. There are no options to alter this behavior.

## ENVIRONMENT VARIABLES

### AS\_SYSLIB

A colon-delimited list of PDS data set names (fully qualified and unquoted) that are to be considered the "base" macro libraries to be used during assembly.

### AS\_PARMS

A comma-delimited list of HLASM options.

### TPFLDR\_OBJLIB

Whenever the **-O** ("big OH") option is present without an argument, the *leftmost* DSN in this list is accessed as the output PDS.

### TPFLDR\_ADATA

A colon-delimited list of PDS data set names, to the leftmost of which **as** will write the debugger information file as a member when the **-g** option is present on the command line. **This environment variable is new as of March 2001 and must not be used unless you are at TPF level PUT 13 or higher.**

## SEE ALSO

c89(1), ld(1), mkmaclib(1), tpfldr(1), tpf-cross(7)

## WHO TO BLAME

Jim Tison, jtison@us.ibm.com

## NAME

fi xedrec – Convert linefeed delimited fi le to fi xed length or vice versa

## SYNOPSIS

```
fixedrec [ -b ] [ -ooutput_fi lename ] [ input_fi lename ]  
[ ...input_fi lename ]
```

## DESCRIPTION

**fixedrec** converts a fi le with fi xed 80-byte records without line feeds to a normal hierarchical fi le system (HFS) text fi le (variable length, line feed-delimited, trailing spaces suppressed). **fixedrec** also can perform the inverse of this conversion.

## OPTIONS

The following command line options may be passed to **fixedrec**:

**-b** Break fi xed-length 80-byte records without line feeds into normal HFS text. The default, in the absence of this option is to remove linefeeds from a normal HFS text fi le and pad each line with trailing spaces to 80 bytes as necessary.

**-o** *output\_*fi lename**  
Write the output to *output\_*fi lename** instead of standard output (stdout).

## INPUTS

Normally, **fixedrec** reads its input from standard input (stdin). However, whenever one or more *input\_*fi lename** tokens is present on the command line, input is read from these fi les as they appear in left-to-right order, and any contents of standard input are ignored.

## OUTPUTS

Normally, **fixedrec** writes its output to standard output (stdout). However, presence of the **-o** option directs that the output will be written to the fi le named as its argument.

Note that all output is written only to a single fi le, irrespective of the number of input fi les given.

## NOTES

### [1] Why does this command exist?

This command was written to assist those users of *ld(1)* who may need to edit HFS-resident defi - nition side deck (DSD) fi les. Although these fi les are written on HFS, they retain the internal RECFM=F, LRECL=80 format required of DSD fi les resident on the MVS cataloged fi le system.

Owing to interaction between the *Program Management Binder*, *ld(1)*, contents of the DSD fi le's IMPORT record, and the possible HFS-residence of output DSD fi les, it is entirely possible that the output DSD may need to be edited on HFS to change the imported module name.

**fixedrec** allows the DSD fi le's format to be changed so that it can be edited with a standard OS/390 UNIX text editor, and then changed back into the RECFM=F, LRECL=80 format that the Binder expects.

### [2] Use on text files only!!

Stream I/O methods that are used on the input fi le are sensitive to bytes containing the line feed value. Such values may occur at random in fi les with binary content. We can *guarantee* unexpected results when **fixedrec** is used with binary fi les.

## SEE ALSO

*ld(1)*, *sed(1)*, *vi(1)*

## WHO TO BLAME

Jim Tison, jtison@us.ibm.com

## NAME

ld – Linkedit a TPF load module (DLM,DLL,LLM) or CP

## SYNOPSIS

```
ld [ -oHFS fi lename ] [ -OPDS fi lename ] [ -v ] [ -M ]
  [ -h ] [ --dln | --dll | --shared | --llm | --cp ]
  [ -w ] [ --help ] [ --version ] [ --syslib=PDS_list ]
  [ -Woptions ] [ -Woptions ] [ --objlib=PDS_list ]
  [ --sidedeck=PDS_list ] [ --import=PDS_list ] [ -p ]
  [ --export=PDS_list ] [ --entry=entrypoint_name ]
  [ object_file_name [...] object_file_names ]
  [ --hversion=header_version ] [ -Ldirectory ]
  [ -lname ] [ --id=tag ] [ --rebind=dllldlmlm ]
  [ HFS sidedeck references ] [ HFS archive references ]
```

## DESCRIPTION

ld calls the standard OS/390 prelinker and/or linkage editor program products to create a *TPF load module* or to link a TPF Control Program.

## INPUTS

### object\_file\_name

The name of at least one, but possibly more *object code files* that are to be linked together to compose your load module. HFS-resident object files end with the file suffix *.o*. Object file names without any suffix will be considered to be members of some PDS in the *--objlib* concatenation (see **OPTIONS**, below). Object file names that exist on HFS and bear suffixes other than *.o*, *.x*, or *.a* will be read into the input as if they were text files composed of 80-byte records; this option is useful for calling in text files composed of linkage editor directives, if needed, particularly as may be needed for linking the TPF control program.

### HFS sidedeck references

These are HFS files produced as the result of building a DLL that contain references to functions contained therein. These files always have the suffix *.x*.

### HFS archive references

These are HFS *archive* files, which is approximately the HFS analog to a PDS. These may be used to contain object files *only*, and may be used to resolve autocall references.

## OPTIONS

The following command line options may be passed to *ld*:

### -oHFS *fi lename*

Create the load module output as an HFS file with the name *HFS\_fi lename*.

### -OPDS *fi lename*

Create the load module output as a member of a PDS. *PDS\_fi lename* may be presented either a simple member name, or as a DSN(MEMBER) name. If the simple member name format is used, the DSN into which the member is stored will be obtained from the TPF\_LD\_SYSLMOD environment variable.

**NOTE: At least one of -o or -O must be specified and they are mutually exclusive.**

Note also that the *HFS\_fi lename* or the PDS membername as given causes a linkage editor NAME directive to be generated. This will also affect the name of the DLL to be loaded in any side deck output created by the *--dll* option.

### -v, --version

Display the current version/release number of *ld*.

### -M

Cause a load map to be generated in the prelinker and linkage editor phases.

- h, --help**  
Display an on-screen short help for command line format.
- dlm** Cause the load module output to be created as a TPF DLM. This is the default if **--dlm**, **--llm**, **--dll**, **--shared**, or **--cp** is not provided on the command line.
- dll, --shared**  
Cause the load module output to be created as a TPF DLL.
- llm** Cause the load module output to be created as a TPF LLM.
- cp** Cause the load module output to be created as a TPF control program. For this type of output only, the prelinker is not called.
- w** Force any return code from the Binder from 1 to 4 to be reported as zero. This option is useful when weak external references are present in the final load module and *make* is in use.
- syslib=*PDS list***  
Specify MVS cataloged PDSs as the 'autocall' library. Here, *PDS\_list* may not contain any member name references. More than one DSN may be named in this option; when doing so, delimit the DSNs with a colon (:) and no intervening white space.
- W*options***  
Specify options to be passed to the prelinker phase. See the *OS/390 C/C++ Compiler Users Guide* for a description of valid prelinker options.
- W*options***  
Specify options to be passed to the binder phase. See *OS/390 Program Management* for a description of valid binder options.
- objlib=*PDS list***  
Specify MVS cataloged PDSs as sources of object library INCLUDEs. Here, *PDS\_list* may not contain any member name references. More than one DSN may be named in this option; when doing so, delimit the DSNs with a colon (:) and no intervening white space. Unsuffixed file references given on the command line are included from this DSN concatenation.
- entry=*entrypoint\_name***  
Alter the default entry point name to some other name. **NOTE: Do not use this option on TPF load modules.**
- sidedeck=*PDS\_memberlist***  
Specify an MVS PDS to contain the definition side decks required to dynamically link to functions present in a DLL. The PDS DSN *and* the members from it you need to reference are required. More than one such file specification may be given as the operand to this option; separate them with a colon (:) and no intervening white space.
- import=*PDS\_memberlist***  
This option is an alias for **--sidedeck**.
- export=*fi lename***  
This option can be used with the **--dll** option to rename the output *side deck definition file* to something other than the name of the module plus *.x*. *fi lename* may also name an existing MVS PDS DSN (no member name), into which the side deck file will be stored. Note carefully that if an MVS DSN is given, it *must exist* - **ld** will never allocate a new MVS cataloged file system file.
- p** Causes the run to consist of the *prelinker* only. This option is not valid with the **--cp** option, and may be used with only the **-o** (HFS file) option for output.
- hversion=*header\_version***  
Causes the inclusion of the indicated version of the *load module header* into the final module, the actual name of which depends upon the type of load module being built. Default for this value is "40". If entered, the value for *header\_version* must be exactly two characters long. If you need a non-default value for this option, your TPF administration staff should be able to tell you.

**-L directory**

Adds *directory* to the default search order for archive libraries specified with the **-I** option. MVS cataloged data set names *cannot* be used here.

**-lname** Adds *libname.a* to the list of UNIX *archive libraries* that the prelinker (or binder, in the case of **--cp**) will search for external reference definitions. MVS cataloged data set names cannot be used here. Note also that the token "lib" will be prepended to *name*, and that the token ".a" will be concatenated to *name* to locate the actual file name. Archive libraries specified with this option should reside in one of the directories specified with the **-L** option.

**--id=tag** Adds an extra CSECT to the final load module, containing the *tag* value, in a format suitable for the TPF online ZDMAP display. IBM uses this to insert the APAR number into load modules it ships. When present, *tag* must be less than eight (8) characters long.

**--rebind=dllldlmllm**

The **--rebind=** switch is for use in applying IBM APARs *only*. Explicit instructions in its use will accompany the APAR and should be followed *exactly*.

## MVS CATALOGED FILESYSTEM NAMES

*ld* does **not** use the standard OS/390 UNIX format to reference a traditional MVS cataloged data set name. Instead, MVS data set names are presented without preceding double slashes (*//*) and without surrounding apostrophes. All MVS data set names presented on the command line must be fully qualified - TSO qualification rules are *not* honored within this utility; nor may UNIX "globbing" features (the use of "?" and "\*" characters in filenames to search on all filenames) be used with MVS dataset names.

Whenever a judgment is to be made about the existence of an MVS data set, its catalog information is sought. All MVS data sets used for output *must exist*; **ld** will never allocate a new cataloged MVS data set. If a collision (identically named files on the HFS and MVS file systems) occurs, the HFS file will prevail unless context implies otherwise.

By imposing these rules, we ensure that command line usage of MVS data sets is as painless as possible by alleviating the need for escape characters or quoting wherever possible; and we further prevent you from inadvertently polluting your MVS catalog high-level qualifier with unexpected datasets and the RACF or SMS "surprises" that might result.

MVS cataloged filesystem names may appear in uppercase, lowercase, or mixed case on the command line -- all DSNs are automatically folded to uppercase by the system.

## PDS NAMES

Wherever a *PDS name* is called for as an option, we mean the fully qualified DSN as described above, without any member specification (name enclosed in parenthesis pair).

## PDS LIST

Wherever a *PDS list* is called for as an option, we mean one or more PDS names as described previously. When more than one PDS name is present in the list, it is to be delimited from its predecessor by a colon and no intervening spaces.

## PDS MEMBER LIST

Whenever a *PDS member list* is called for as an option, here we mean a fully qualified PDS DSN, followed immediately by an open parenthesis, followed by one or more PDS member names and a closing parenthesis. When more than one PDS member name is present, it is to be delimited from its predecessor by a comma and no intervening spaces. These items may also be considered "lists of PDS member lists", in which case the DSN and member specifications are colon-delimited as if they were normal PDS lists without members.

## HOW IT WORKS (Load Modules Only)

For TPF load modules (DLM, LLM, and DLL), a composite object file is created from the files named on the command line. The appropriate header for the type of load module begins the composite object file. When object files are called for, these are copied into the composite object. PDS members named on the

command line are read in from DD name OBJLIB. Next, any side deck references are read into the composite object in a similar manner. Finally, any UNIX archive library references are placed at the end of the composite object file so they can be used to resolve all outstanding unresolved references before autocall processing.

The composite object file is then passed to the prelinker as input. Its output (DD name SYSMOD) is stored as an intermediate HFS file in the current working directory. All pre-existing DD name allocations are released once the prelinker has completed, and a minimal set of compiler run-time libraries is allocated as DD name SYSLIB for the binder. Once the binder is completed, the existing DD allocations are freed and the HFS-resident intermediate prelinker output file is deleted.

## HOW IT WORKS (TPF Control Program Link)

Much like the TPF load module case, a composite object file is built in the current working directory on HFS, except that no header object files are necessary. This composite object is then passed straight to the binder without calling the prelinker. There are no "hidden" DD name allocations in this type of link as opposed to what happens when the prelinker is called for linking TPF load modules, nor is there any default content in the composite object file other than that inferred by the command line.

Users of the `--cp` option frequently need to pass `ALIGN` and `PAGE` directives to the binder. You can do this by creating either a member of a PDS (`RECFM=F, LRECL=80`) or an HFS text file that contains the directives. If you choose to create an HFS file, provide some suffix on the filename that is not `.o`, `.x`, or `.a`. `ld` will copy such files directly into the composite object file, and will use record padding to ensure that all records extend to the full 80 bytes required by the binder.

## ENVIRONMENT VARIABLES

### TPF\_LD\_SYSLMOD

This environment variable should name a PDS into which the member named by the `-O` option, if present, is written. This environment variable is not interrogated if the `-o` option is present on the command line, or the `-O` option is presented in `DSN.NAME(MEMBER)` format.

### TPF\_LD\_SYSLIB

Substitutes for the `--syslib=` command line option. This environment variable should name a colon-delimited list of PDS names which will be used for autocall resolution. The contents of this environment variable are *cumulative* with the `--syslib` option, with the data sets named on the command line searched *before* those named in `TPF_LD_SYSLIB`.

### TPF\_LD\_OBJLIB

Substitutes for the `--objlib=` command line option. This environment variable should name a colon-delimited list of PDS names which will be used for PDS-resident object files. The contents of this environment variable are *cumulative* with the `--objlib` option, with the datasets named on the command line searched *before* those named in `TPF_LD_OBJLIB`.

### TPF\_LD\_SIDEDECK

Substitutes for the `--sidedeck=` command line option. This environment variable should name a colon-delimited list of PDS names *and member names* which will be used for PDS-resident definition side deck (DSD) files. The contents of this environment variable are *cumulative* with the `--sidedeck` option, with the data sets named on the command line searched *before* those named in `TPF_LD_SIDEDECK`.

## NOTES

### [1] Order of UNIX library processing

UNIX **ar** archive libraries are always referred to from the *end* of the composite object file, with `.a` files named explicitly on the command line preceding any named with the `-L` and `-I` switches. This guarantees that all outstanding external references from the object files will be resolved from the archive libraries before autocall processing. Any archive libraries introduced with the `-L` and `-I` switches will be referred to after any explicitly named `.a` files and in the left-to-right order in which their `-I` references occurred.

## [2] Sidedeck creation in MVS PDS datasets

When option **-O** (big-OH) is present on **--dll** processing, an attempt will be made to locate an MVS data set named similarly to the PDS into which the load module will be written, except that the last qualifier will be **.EXP**. If this data set is not located, and no **--export** option is present, **ld** will terminate without processing anything. If the **--export** option names an HFS dataset or a non-existing MVS cataloged data set, the side deck file will be written to the HFS file system, not the MVS cataloged file system.

## [3] Use of the current working directory

The **ld** user must always have write permission on the current working directory, even if outputs are specified to be written in PDS data sets or other HFS directories. Intermediate files are written to the current working directory.

## [4] Names and the prelinker/binder NAME directive

The prelinker, as it creates the *definition side deck* (DSD) file for DLLs, must know the name of the module it is creating. The way it knows about the name of the module is to have it identified in a NAME directive statement. **ld** derives this name from the **-o** or **-O** option.

The operand to the generated NAME directive is the name of the module, as literally presented -- in other words, the PDS member name or the HFS file name to which it is to be written. Because PDS member names must be in uppercase, the NAME directive operand is also folded to uppercase if necessary. However, HFS file names are taken literally in the case in which they were provided on the command line.

When a DLL is created, the prelinker creates the DSD. The module name is taken directly from the NAME directive. As such, HFS module names presented in mixed case or lowercase are used literally in the DSD IMPORT content -- for TPF usage, this is not what we want. Unfortunately, folding the operand of the NAME directive into uppercase isn't a good thing to do to HFS outputs either -- the binder will take the NAME statement literally and override any DD name definition for SYSLMOD, and write the file to HFS in uppercase letters. To preserve the integrity of HFS filenames, the literal case of the module name as presented on the command line is also preserved. This means that the DSD may have to be manually edited to insert the proper TPF segment name (6 characters, including version, in uppercase) instead of the mixed case name of indeterminate length inserted by the prelinker.

## [5] Editing HFS sidedeck files

In [4], we presented a case where a sidedeck file will probably have to be edited. This becomes somewhat difficult under UNIX System Services because of the format of a sidedeck file.

The sidedeck file on HFS is a text file that contains no linefeeds. Each "logical line" in this format is 80 characters long, much as if the file were resident in a MVS cataloged data set with DCB attributes of RECFM=F and LRECL=80. Standard UNIX editors such as *vi*, *e*, and *lpex* cannot handle this format correctly -- what they will display is one seemingly "long" record.

If the term to be "substituted out" is equal to the length of the term to be "substituted in", *sed(1)* is a viable option for this kind of use. However, for those who can't stomach *sed* or must have differing string lengths, the TPF cross-development environment provides the **fi xedrec** utility.

We suggest the following use of **fi xedrec** with sidedecks, where the sidedeck file to be edited is called "mysds.x":

```
fi xedrec -b mysds.x | sed -e 's/qzz1/QZZ140/g' | fi xedrec -o mysds.x
```

This example uses *sed* to substitute the string "QZZ140" for the string "qzz1", and represents a frequent case where an HFS filename target of the IMPORT statement would have to be changed to a long TPF module name, in uppercase, with a following version number.

Instead of using *sed* as shown above, you could use the HFS editor of your choice as long as you follow these steps, which the example shows:

(a) Use **fi xedrec -b** to trim trailing blanks from each incoming 80-byte record and append linefeeds. Note that the output from *fi xedrec* should not be written to the original file. In the example, we pipe the output directly to *sed*, substituting for step (b), which follows.

(b) Edit the file, making the necessary changes.

(c) Use **fi xedrec -o** to simultaneously re-expand each record to 80 bytes and remove all linefeeds and rename the output with the *-o* switch. Note that the input was piped in from the output of *sed* in the example.

#### [6] Binder options

In non-CP links (DLM, DLL, LLM), the following options are fed by **ld** to the binder following the completion of the prelinker phase:

```
TERM,AMODE=31,RMODE=ANY,ALIASES=NO,REUS=RENT,CASE=MIXED,UP-  
CASE=NO
```

In CP links, the following options are fed by **ld** to the binder/linkage editor:

```
TERM,AMODE=24,RMODE=24,LET,DCBS,CASE=MIXED,COMPAT(PM2)
```

There is no prelinker phase in *--cp* links.

Any of these default options may be reversed or enhanced by using the **-Wl** option.

#### SEE ALSO

*libi(1)*, *c89(1)*, *as(1)*, *lmcp(1)*, *fi xedrec(1)*, *tpf-cross(7)*

#### WHO TO BLAME

Jim Tison, [jtison@us.ibm.com](mailto:jtison@us.ibm.com)

## NAME

`libi` – Generate LLM stubs and transfer vector

## SYNOPSIS

```
libi [ -v ] [ -bbuildscript_fn ] [ -sstub_directory ]  
[ -xtransfer_vector_fn ] library_name
```

## DESCRIPTION

`libi` is used to create a *transfer vector* object file and a set of *stub* object files.

## OPTIONS

The following command line options may be passed to `libi`:

- v** Causes the *build script* input to `libi` to be echoed as it is processed.
- bbuildscript\_fn**  
Forces `libi` to read *buildscript\_fn* as its input instead of **library\_name.lsc**.
- sstub\_directory**  
Forces `libi` to write its stub object files to *stub\_directory* instead of **library\_name.stubs**.
- xtransfer\_vector\_fn**  
Forces `libi` to write its transfer vector object file to *transfer\_vector\_fn* instead of *library\_name(1:4)xvlibrary\_name(5:6).o*.

## HOW IT WORKS

`libi` examines the build script input and generates a set of object files; one for the *transfer vector* and one *stub* object file for each library function named in the build script.

The *stub* object files merely contain linkage code to call the *transfer vector* with the correct library routine number. These stub files are used by programs that refer to the present LLM at run time, and should be statically linked by any caller needing a function in the LLM.

## HOW DO I USE THESE FILES?

The *transfer vector* should be statically linked into your LLM as the **first** object; the remaining objects from your compilations or assemblies (that is, those object files that represent the actual runtime function) can follow in any order. When `libi` runs to completion without the `-x` option, this object file will be created and named by taking the first four characters of the *library name* (the TPF segment name for the LLM), concatenating **xv** to it, then concatenating the last two characters of the *library name* to that, and then appending the `.o` file suffix. NOTE: when not using the `-x` option, it is important that your *library name* is exactly six characters long and composed as described.

You can use `ld` to link the transfer vector object file into your LLM; it should be listed first on the command line list of object files.

Use of the *stub object files* is straightforward; link the appropriate stubs into a calling program. **Disposition** of the stub object files, however, is up to each programmer. These may be copied to a PDS with the TSO `OGETX` command, or placed into a UNIX *archive file* with the `ar` command so they can be referred to during later link-editing operations.

In all cases, `libi` writes the stub object files to a subdirectory of the current working directory, whose name is either what was passed by the `-s` switch or the library name + `".stubs"`. All files written to the *stub directory* will have the suffix `".o"`, and will be the lowercase representation of the function name encountered in the library script. For example, function FOO named in library script CABCU0 will cause a file called `./cabcu0/foo.o` to exist.

## SEE ALSO

`ld(1)`, `TSO OGETX`, `ar(1)`, `libi-script(5)`, `tpf-cross(7)`

## **WHO TO BLAME**

Jim Tison, jtison@us.ibm.com

## FILE DESCRIPTION

The **libi script** file is used as the only input to the *libi(1)* command, describing the function numbers to be assigned to named entry points in a TPF library load module (LLM).

## FILE NAMING REQUIREMENTS

The **libi script** file should have a file suffix of .lsc. If it has another suffix, the file will need to be fully identified to the *libi(1)* command with the **-f** switch. The part of the file name preceding the suffix should match the name of the library being built.

## FILE FORMAT

The **libi script** file is a stream of newline-delimited EBCDIC text. Each line in the file begins with one of the following three tokens:

#        The hash character indicates that the contents of this line from its point of occurrence rightward are *comments* and do not participate in stub or transfer vector generation.

### @libid(*n,name*)

The *@libid* directive identifies the LLM to libi. The TPF 4-character *segment name* should appear instead of the *name* argument, and the *n* argument should be a unique LLM number in your system.

### @libfun(*n,name*)

The *@libfun* directive identifies an LLM member function for which a transfer vector entry and a corresponding stub object file are to be generated. *name* should name the function entry point, and *n* should be a **unique** function number in the LLM and this script file.

## RESTRICTIONS

There must be only one *@libid()* line per **libi script** file, and there should be as many *@libfun()* lines as there are function members of the LLM, each bearing a *unique* function number.

## SEE ALSO

ld(1), libi-script(5), tpf-cross(7)

## WHO TO BLAME

Jim Tison, jtison@us.ibm.com

## NAME

lmcp - Copy TPF load module from HFS to PDS

## SYNOPSIS

```
lmcp    [ -h ] [ -v ] [ -rnew_membername ]  
        [ -e[entrypoint_name] ] HFS_loadmod_fn  
        [ MVS_loadmod_DSN ]
```

## DESCRIPTION

**lmcp** copies a TPF load module (DLM, LLM, or DLL) from the Hierarchical File System to a traditional cataloged OS/390 partitioned data set (PDS) load library.

Given that the formats of HFS and PDS load modules are different, this cannot be accomplished with a "read-record, write-record" type of utility, such as **cp** or TSO **OGET**. Instead, the OS/390 Program Management Binder (IEWBLINK) is called to perform the re-edit into the PDS target DSN.

May also be used with conditions (see the **-e** option description, which follows) to copy non-TPF HFS load modules to PDS/Es.

*lmcp* only copies modules **from HFS to** the traditional MVS cataloged file system. There is no support for movement in the opposite direction (see *pcml(1)*).

## OPTIONS

- h** Displays a brief usage syntax message on the console. No additional processing occurs.
- v** Displays binder output on the console.
- rnew\_membername**  
Causes the name of the newly created (or replaced) PDS member to be *new\_membername*. Default behavior to name the member from the first 6 characters of its HFS file name stem.
- e[entrypoint\_name]**  
Specifies the entry point name of the load module; not necessary for TPF modules. If *entrypoint\_name* is not given when this switch is present, **CEESTART** is assumed. Otherwise the name is passed literally (without folding to upper case) as the entrypoint. Use of this option is the only way to get a TPF control program load module processed by **lmcp**.

## INPUTS

### HFS\_loadmod

The name of an existing HFS-resident TPF load module (DLM, LLM, or DLL). May be a non-TPF load module if the **-e** option is given.

[ MVS\_loadmod\_DSN ]

The DSN of the MVS PDS load library (PDS) to which you want *HFS\_loadmod\_fn* copied. Note that the specification of the MVS DSN must *not* be enclosed in apostrophes, and it also must be fully qualified. This specification must **not** specify a member name in parentheses.

## LOAD MODULES OTHER THAN DLM/LLM/DLL OR TPF CP

Use of the **-e** option causes the entry point name to be specified by command line input (or default) instead of dynamically testing the module for its entry point name as we do for TPF modules. In this manner, we can copy non-TPF load modules from HFS to a **PDS/E** -- this is sometimes convenient for those developing offline support utilities. When copying such a module using **lmcp**, we need to keep in mind that the standard HFS load module is bound (link-edited) with the *Program Management Binder*, and not the prelinker invoked for TPF load modules. Programs bound with only the Binder **must be copied to a PDSE**, not a PDS -- this is a Binder restriction. Attempting to copy a non-TPF load module to a regular PDS will cause the bind process to fail.

## FACTORS FOR FAILURE

If the **lmcp** command fails, run it again with the **-v** command line option. This will cause the Binder output (map) to be displayed on your console and will pinpoint the reason for failure. The following are common

reasons for failure:

**No load module to copy.**

You have not created an HFS load module. TPF users need to run **ld(1)**; or non-TPF users need to run **c89**.

**Linkage symbols missing.**

You ran your original link-edit of the HFS load module with the **-Wl,EDIT=NO** option, thus stripping out all internal symbols. This is a no-no, because we have to rebind to do the copy.

**Attempting to copy a non-TPF module to a PDS.**

Copy to a PDS/E instead.

**Incorrect entry point name supplied.**

If you used the **-e** command line switch *and* supplied an entry point name, make sure that it is correct and expressed in the proper uppercase or lowercase characters. If you didn't supply an entry point name, remember that the default is **CEESTART**.

**ENVIRONMENT VARIABLES**

**TPF\_LD\_SYSLMOD**

If the *MVS\_loadmod\_DSN* command line token is not present, the data set name present in this environment variable, if any, is used as the DSN target for the copy operation.

**SEE ALSO**

ld(1), c89(1), tpf-cross(7)

**WHO TO BLAME**

Jim Tison, jtison@us.ibm.com

## NAME

mkmaclib – Create a PDS from HFS macros

## SYNOPSIS

```
mkmaclib -d dataset_name
          macro_spec [...macro_specs ]
```

## DESCRIPTION

**mkmaclib** creates a new MVS cataloged partitioned data set from hierarchical file system (HFS) file contents.

## SWITCHES

**-d** *dataset\_name*

This switch defines the name of the PDS that is to be created from the following HFS macro specifications. If *dataset\_name* is already cataloged, it will be deleted and uncataloged before creating the new one.

## INPUTS

*macro\_spec*

One or more *macro\_spec*ifications may appear. These are relative or absolute paths to either HFS files or directories. When a file is named, it must have the suffix .mac or .cpy. Whenever a directory is named, files in it that have the suffix .mac or .cpy are accessed and become members of the PDS. At least one HFS macro specification must appear, and as many macro specifications as necessary may appear.

## PDS MEMBER NAMES

Each HFS file is stripped down to its **stem**, which is considered to be everything between the end of the path prefix (on the left) and the first dot to its right. For example, the stem name for /u/myuid/myproject/dctpfx.mac is dctpfx. The only ways in which this name will be manipulated will be to translate underscore characters into @ characters, and to fold to uppercase. If there is any other violation of MVS PDS member naming rules, the HFS file will not be considered for inclusion in the final maclib, and an error message identifying the HFS file in error will be displayed on stdout.

## OUTPUT

There are two outputs from **mkmaclib**: console messages and the created PDS.

The console messages normally consist of a composition report stating the member names and the corresponding HFS file of origin. In certain error conditions, such as HFS files that are unreadable, of length zero, or have stem names that do not convert into PDS member names, the composition report will be prefaced by appropriate messages.

The output PDS itself is created with DISP=NEW,CATALOG, UNIT=SYSDA, with enough space in the primary extent to hold all anticipated data. At present, the block size is 27920, which is the most efficient usage possible of track surface with 3390 DASD geometry. The secondary extents of this PDS are set for 5% of the primary extent plus 1 track. While it is likely that the primary extent may contain some additional "slack" space, users may do well to keep the original source of their macros around on HFS, and merely re-execute **mkmaclib** rather than editing PDS members by other means; this will prevent the PDS from growing into secondary extents. This is important because the total number of used extents in a PDS concatenation is limited to 120, and PDSs with used secondary extents limit the total number of data sets that may appear in any single concatenation.

## LIMITS AND CAVEATS

As of this writing, we will limit macro libraries created by **mkmaclib** to 1,300 members.

**mkmaclib** does not and cannot anticipate SMS or RACF limits imposed at any installation. Such limits will usually be noticed during dynamic DD name allocation, however. In cases where allocation is prohibited by SMS or RACF and successfully detected, **mkmaclib** will stop. However, if the **-d** switch named an existing data set, it will already have been uncataloged and deleted by the time we detect the allocation error. We therefore suggest extreme caution when re-creating existing data sets that multiple people may depend on.

The internal mechanism for resolving potential namespace conflicts is identical to that used in *as(1)*. Refer to the *as(1)* manual page for details.

**ATTENTION:** **mkmaclib** is optimized for 3390 DASD geometry. Attempting to use **mkmaclib** on 3380 (or earlier) DASD units may waste large amounts of DASD surface.

**SEE ALSO**

*as(1)*, *tpf-cross(7)*

**WHO TO BLAME**

Jim Tison, [jtison@us.ibm.com](mailto:jtison@us.ibm.com)

## NAME

pcml - PDS Copy Loadmodule to HFS

## SYNOPSIS

```
pcml [ -h ] [ -v ] [ -r [path/]fi lename ]  
      MVS_loadmod_DSN(member)
```

## DESCRIPTION

**pcml** copies a PDS-resident load module (DLM, LLM, or DLL) to the OS/390 UNIX Hierarchical File System (HFS).

Given that the formats of HFS and PDS load modules are different, this cannot be accomplished with a "read-record, write-record" type of utility, such as **cp** or TSO **OGET**. Instead, the OS/390 Program Management Binder (IEWBLINK) is called to rebind the PDS source load module into the HFS.

Default behavior is to write the load module into a regular file (non executable attributes) in the current working directory, named identically to the source PDS membername, folded to lowercase. Alternatively, the **-r** option may be used to specify an alternate filename and/or path.

This function is the inverse of the **lmcp**(1) command, whose name is reversed. It is expected that **pcml** will be a low-usage command, since **ld**(1) is capable of building the original load module directly onto the HFS, and that such capability is preferable for reasons of **make**(1) compatibility.

## OPTIONS

- h** Displays a brief usage syntax message on the console. No additional processing occurs.
- v** Displays Binder output on the console.
- r***[pathname/]fi lename*  
Overrides the default HFS filename (and possibly path) of the load module output. Default behavior is to write a file to the current working directory - its filename will be equal to the source PDS member name in lower case.

## INPUTS

**MVS\_loadmod\_DSN(membername)**

The name of an existing PDS-resident load module (DLM, LLM, or DLL).

## SEE ALSO

ld(1), lmcp(1)

## WHO TO BLAME

Jim Tison, jtison@us.ibm.com

## NAME

tpfldr – Create a TPF loadset

## SYNOPSIS

```
tpfldr [ -ssaltbl_dsn ] [ -oobjlib_dsn_list ]  
[ -zclmsize ] [ -tOLDR|TLDR ] [ -lloadmod_dsn ]  
[ -acp_dsn ] [ -ggds_dsn | -vdest.uid ]  
[ --hfsmod=HFS_fi lename_list ]  
[ --hfsobj=HFS_o_list ]  
[ --hfsadata=HFS_adata_directory_list ] input_hfs_fi lename
```

## INPUTS

**tpfldr** reads its *loader control statements* (formerly known to us in the MVS JCL version as DD name SYSIN) from the *input\_hfs\_fi lename* expressed on the command line. This file must contain nothing other than loader control statements.

## OPTIONS

*-s saltbl\_dsn*

The fully qualified name (see **MVS DATASET NAMES**) of the SAL table PDS that you want to use for external symbol resolution.

*-o objlib\_dsn\_list*

A colon-delimited list of PDS DSNs containing unlinked object code to be included as standard TPF E-type segments in the loadset.

*-z clmsize*

A number (without commas or other punctuation) that represents the combined load module size. If this option is missing, **tpfldr** uses a default of 15000000 (yes, 15 MB).

*-t OLDR|TLDR*

The **type** of offline load requested; either the constant OLDR or TLDR. OLDR is the default when this option is omitted.

*-l loadmod\_dsn\_list*

A colon-delimited list of PDS DSNs containing linked TPF load modules (DLM, DLL, LLM) for inclusion in the loadset.

*--hfsmod=HFS\_fi lename\_list*

A colon-delimited list of HFS file names containing linked TPF load modules for inclusion in the loadset. This option may appear more than once in any single command line.

*-a cp\_dsn*

(Applicable only when **-tTLDR**; ignored otherwise if present.) The name of the MVS cataloged PDS that contains the link-edited TPF control program. Analogous to DD ALDRCP.

*-g gds\_dsn* | *-v dest.uid*

Specifies the final destination of the completed loadset. The **-g** option writes the output to the TPF general data set (GDS) named as its argument. The **-v** option causes the completed loadset to be punched over RSCS to the destination machine named by *dest* and the user ID on that machine named by *uid*. The dot delimiting the destination and user ID is required. See **USAGE NOTES**[1] for a more thorough discussion of this option. At least one of **-g** or **-v** (or their surrogate environment variables) must be present.

*--hfsobj=HFS\_o\_list*

Specifies that the colon-delimited list of file names identified by *HFS\_o\_list* represents one or more HFS-resident object files (.o suffix) that are to be included in the DD name OBJDD (when **-t TLDR**) or OBJLIB (**-t OLDR**) ahead of any MVS cataloged file system-resident PDS members.

*--hfsadata=HFS\_adata\_directory\_list*

The colon-delimited list of HFS directories (represented by *HFS\_adata\_directory\_list*) is searched for HFS files patterned like \*.ADATA. If any are found, they are considered to be sources of VisualAge TPF Assembler Debugger symbolic information that take priority over any

sources found in TPFLDR\_ADATA. Any listed subdirectory that does not contain \*.ADATA files is considered a fatal error. **NOTE: This option has no effect until TPF PUT13.**

## MVS DATASET NAMES

This command, like `ld(1)`, allows you to enter fully qualified MVS cataloged PDS names without surrounding them with apostrophes or prefacing them with `"/"`. Dataset names so entered may be in either uppercase or lowercase. Whenever a PDS name is called for, no *membername* or parentheses are permitted.

## HOW IT WORKS

`tpflr` is an OS/390 UNIX shell wrapper for the `TPFLDR40` program. Because no MVS load library containing `TPFLDR40` is present in the default STEPLIB, nor in the LPA or link list, users of `tpflr` *must* set the environment variable **STEPLIB** to contain the DSN of the MVS load library in which it resides (see **USAGE NOTES**[2] and **ENVIRONMENT VARIABLES**). `tpflr` performs dynamic DD allocations for all required datasets, and then calls `TPFLDR40` using standard MVS semantics.

`tpflr` uses all contents of MVS cataloged data sets without alteration of any kind. When the `--hfsobj=` option is given, a temporary PDS is created with copies of the .o files as members. When the `--hfsadata=` option is given, a temporary PDS may be created with .ADATA members of such directories, as needed. When the `--hfsmod` option is given, the Program Management Binder will be called to create a temporary PDS containing all of the members specified as its argument(s).

## ENVIRONMENT VARIABLES

Several environment variables are interrogated by `tpflr --` these can be used in lieu of command line options.

### TPFLDR\_SALTB

Contains the name of a single PDS that is to be used as the source for the SAL table. This environment variable is interrogated if the `-s` command line option is not present.

### TPFLDR\_CLMSIZE

Contains a number expressing the combined load module size in bytes. This number is usually quite large, and may not contain commas or white space. This environment variable is interrogated if the `-z` command line option is not present.

### TPFLDR\_OBGLIB

Contains a colon-delimited list of data set names from which unlinked E-type object code is sought. This environment variable is interrogated if the `-o` command line option is not present.

### TPFLDR\_TYPE

Either OLDR or TLDR. This environment variable is interrogated if the `-t` command line option is not present.

### TPFLDR\_GDS

Contains the DSN of the TPF general data set to which you want your loadset written. This environment variable is interrogated if neither the `-g` nor `-v` command line options are present.

### TPFLDR\_ALDRCP

(Applicable to TLDR runs only). Contains a colon-delimited list of MVS cataloged PDSs that are to be considered candidates to house link-edited TPF control program code.

### TPFLDR\_ADATA

Contains a colon-delimited list of MVS cataloged PDSs that are to be considered sources of TPF VisualAge Assembler Debugger file information. **This environment variable has no effect until TPF PUT 13.**

### TPFLDR\_ADATA\_HFS [new in V1.8, October 2002]

Contains a colon-delimited list of HFS directories that are to be considered sources of TPF VisualAge Assembler Debugger file information. See remarks for the `--hfsadata=` command line option. **This environment variable has no effect until TPF PUT 13.**

## TPFLDR\_VERSIONCODE

Intended mainly for CP support programmers; contains the version suffix to append to the string "TPFLDR" when dispatching the MVS EXEC. If this variable is not defined or is empty, the default ("40") is used when TPFLDR\_PROGRAM is not set. When TPFLDR\_PROGRAM is set, this variable is ignored.

## TPFLDR\_PROGRAM

The contents of this environment variable, if present, name the program (STEPLIB member) that is to be called in lieu of TPFLDR40. If it is set to any non-null value, the TPFLDR\_VERSIONCODE variable is ignored. In other words, when TPFLDR\_PROGRAM is set, it is intended to convey the *full* member name of the TPFLDR40 program as installed at your installation.

## STEPLIB

This environment variable, established by OS/390 UNIX System Services, must be set and point to at least one MVS cataloged file system data set name, one of which will hopefully be an MVS load library containing the member TPFLDR40. Users of **tpfltr** who need to modify STEPLIB can do so as follows:

```
export STEPLIB=$STEPLIB:@@STEPLIB@@
```

where @@STEPLIB@@ represents the name of the correct MVS load library as an example. Your site's installation may vary.

## USAGE NOTES

- [1] Output can only be written to one place. **tpfltr** will write either to a TPF general data set (GDS), or punch to an RSCS-connected machine, such as a TPF VM-guested test machine's reader. **tpfltr** *must* have one and only one of these two destinations to write to. The following precedence of command line options and environment variables apply:

- o The **-g** option
- o The **-v** option
- o TPFLDR\_GDS (equivalent to **-g**).

Note that you cannot state a **-v** destination.userid pair in an environment variable. This is deliberate. Using the TPF GDS for output is much more efficient than using the RSCS option. The RSCS option is provided as a backup for those moments when TPF GDS support (or a usable GDS data set) is not available.

- [2] Environment variable STEPLIB will be checked for a non-null state. If it is NULL, **tpfltr** will exit with a non-zero return code and a warning message output to stderr. Each dataset name in the STEPLIB variable will be checked to see if the TPFLDR40 member (or its substitute name, as set by TPFLDR\_PROGRAM or TPFLDR\_VERSIONCODE) is present. If not found, **tpfltr** will exit with a non-zero return code and will output an error message on stderr.
- [3] Because of a conflict in the way the TPF ZOLDR command is coded and the LRECL/BLKSIZE attributes of MVS class 9, **tpfltr** is unable to ship RSCS files with the correct class (9) when using the **-v** option. Instead, to get the correct LRECL and BLKSIZE, MVS class B must be used as the puncher class. This will cause ZOLDR to not recognize the existence of the received file. To remedy this, use the following VM CP command:

```
#CP CH RDR nnnn CL 9
```

from the TPF machine's console, where *nnnn* represents the spool file number of the reader file received. You may then enter a ZOLDR command against the reader file.

## SEE ALSO

as(1), ld(1), tpf-cross(7)

## WHO TO BLAME

Jim Tison, jtison@us.ibm.com