



Transaction Processing Facility

**maketpf**

Brian K. Laferriere

## Topics:

- Introduction
- Solution Overview
- File Syntax and Content
- Tools
- Build Procedures

## What is maketpf?

- **maketpf** is a GNU make-based build solution for assembling, compiling, and linking TPF system and customer application programs in an OS/390 UNIX environment.
- It is based on a TPF-developed schema, which provides for common build rules and options, can handle flexible hfs structures, and consists of both makefiles and UNIX scripts.
- **maketpf** was initially conceived as the build solution for the next TPF release, but is being retrofitted to TPF 41.

## Why are we providing maketpf for TPF 41?

- As a migration path to the next release
- To provide a common build tool set:
  - ▶ Used by both systems and application programmers
  - ▶ Used for both system and development builds
- For consistency with ported code requirements.

## What are our long-term goals?

- Provide a single solution for building both:
  - ▶ TPF system code
  - ▶ Application code
  
- Drive builds from OS/390 UNIX
  
- Obtain source code from the HFS (not PDS)
  
- Replace JCL with makefiles.

## What are the advantages?

- Provides a single source for build options:
  - ▶ Default assemble/compile/link options reside in a common rules file shared by all makefiles
  - ▶ Overriding options are associated to source parts in the makefiles
  
- Can be customized:
  - ▶ Supports a flexible HFS structure
  - ▶ Can be extended as new HFS structures are added.

## What are the advantages? *(continued)*

- Adds support for customer mods:
  - ▶ Provides an hfs solution for customer mods to TPF source to be kept in parallel to TPF-delivered source, simplifying the maintenance process.
  
- Can manage multiple TPF configurations:
  - ▶ Handles TPF configuration-dependent programs
  - ▶ Can use same makefile with different TPF systems
  - ▶ Optimizes build of configuration-independent programs, for multi-system builds.

## What are the advantages? *(continued)*

- Allows HFS structures to be layered
  - ▶ Eliminating need for duplicate copies of system code
  - ▶ Similar to MVS PDS concatenation.
  
- Provides a common build solution
  - ▶ Can be used for both TPF systems and customer applications
  - ▶ Can be used for both development builds and system builds.
  
- Minimizes makefile knowledge required.

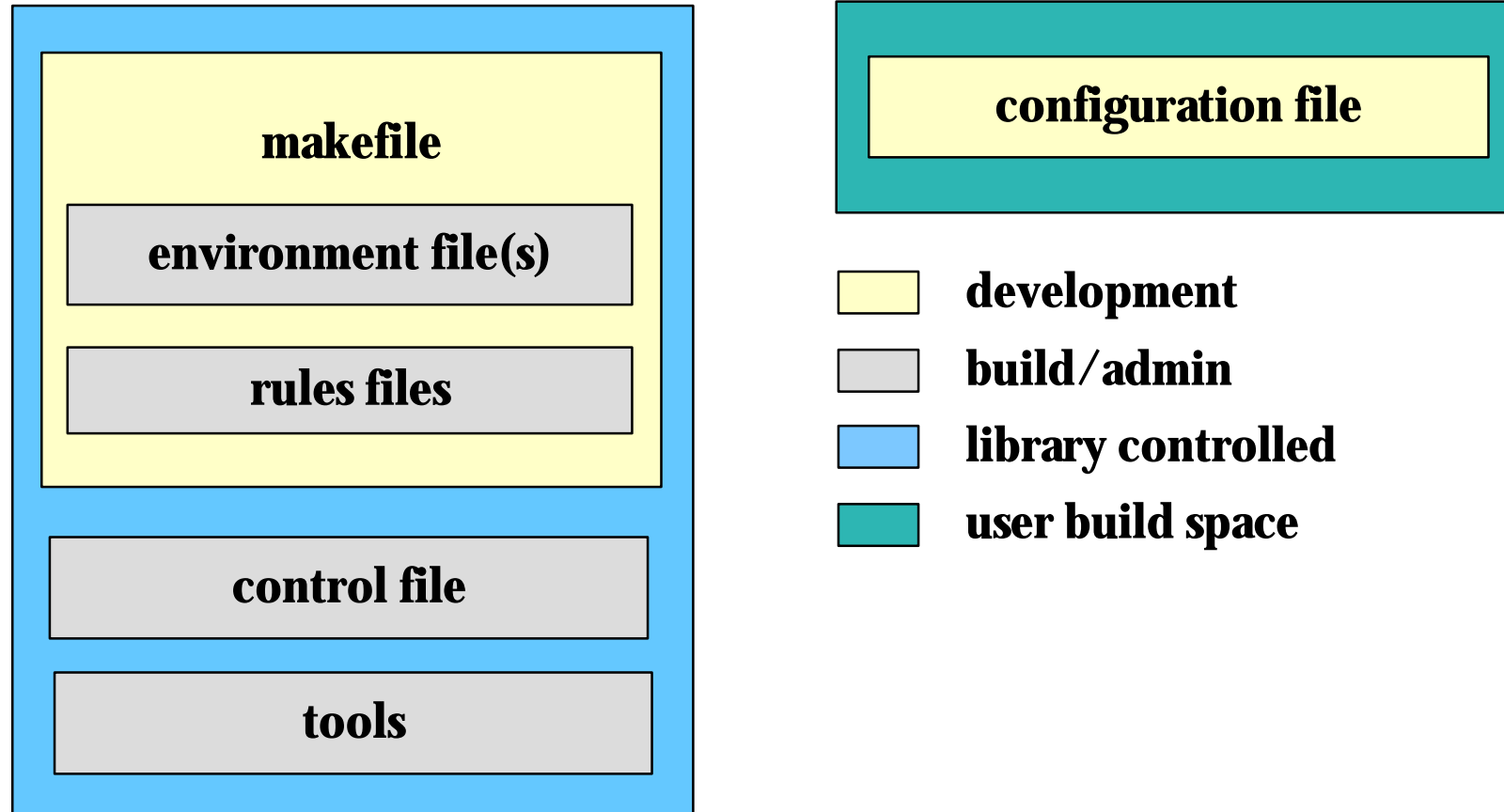
## What can we not do for TPF 41?

- Fully eliminate the need for PDSs:
  - ▶ HLASM requires macro, copy files in PDS
  - ▶ Link utilities on OS/390 UNIX require PDSs for autocall resolution.
  
- Fully replace the SIP process:
  - ▶ The complexity of the TPF 41 build process does not make this feasible.
  - ▶ Only online C modules and assembler real-times could be implemented.

## So what are our short-term goals then?

- Get you started thinking about build migration to the next release.
- Enable you become familiar with the maketpf solution.
- Provide the tools that will enable you to begin building your applications in an hfs environment.
- Obtain feedback from you on what we have provided and what additional requirements we need to address.

## Overview: maketpf



## Overview: makefile

- Defines name and type of the program.
- Defines the source segments belonging to the program.
- Defines the environment(s) required for build.
  - ▶ The first environment listed is used for output.
- Defines any compile, assemble, or link overrides specific to the program.
- Uses GNU make syntax.
- Is intended to be owned and maintained by development.

## Overview: makefile (continued)

- There is one makefile per program.
- The makefile name should be of the format:  
`xxxx.mak`  
where `xxxx` is the program name.
- The output executable will be named:  
`XXXXVV`
- The makefile resides in the source file directories, similar to build scripts.
- The makefile must "include" the common rules file.

## Overview: rules file

- Provides a single source for the compile, assemble, and link rules.
- Defines the default compile, assemble, and link options.
- Handles resolution of actual file pathname using base source file name provided in the makefile from the path information provided in the environment files.
- Is intended to be maintained by the build and admin team.

## Overview: environment file

- Defines logical groupings of programs and their associated hfs (e.g., base rt, debug).
- Maps each hfs directory name to a fixed set of known types (e.g., C Source, Asm Source, Objects) .
- Can be created to support either TPF or application programs.
- Provides support for multiple hfs source trees to be concatenated.
- Are intended to be maintained by the build and admin team.

## Overview: configuration file

- Defines the user's build space.
- Identifies the root names of the hfs source trees to be included in the build.
- Defines the system name and type to build against (used to resolve system-generated source, headers, and macros).
- Provides another means for setting temporary build overrides, eliminating need to change makefiles.

## Overview: control file

- Defines the list of programs that can be processed by maketpf.
- Provides a correspondence table between the program name and the associated makefile.
- Defines the build order, system designations, functions switches, etc. applicable to each program.

## Overview: tools

- **maketpf**
  - ▶ A korn shell script used to set up and run a single maketpf format makefile.
- **bldtpf**
  - ▶ A korn shell script used to set up and run multiple maketpf format makefiles listed in a control file.
- **maketpf.bsc.convert**
  - ▶ A korn shell script used to convert a build script to maketpf format makefile.

## Makefile: Program Content Variables

VARIABLE	DESCRIPTION
CSO	C program name.
CSO_TYPE	C program type: DLL, DLM, LLM
LIBS	List of DSDs needed to resolve external references at link time.
maketpf_env	List of environments required to build the executable. The first environment listed is considered the owning environment. If a TPF environment file is first, the first TPF_ROOT specified is used as the output hfs. If an application environment file is first, the first APPL_ROOT specified is used as the output hfs.

## Makefile: Source Segment Variables

VARIABLE	DESCRIPTION
CC_SRC	List of source segment names to be built using the <b>c89</b> command. Currently files named with suffix <b>.c</b> are supported.
CPP_SRC	List of source segment names to be built using the <b>cxx</b> command. Currently files name with suffix <b>.cpp</b> are supported.
ASM_SRC	List of source segment names to be assembled using the <b>as</b> command. Currently files named with suffix <b>.asm</b> are supported.
OCO_OBJ	List of OCO objects to be included in the module.

## Makefile: Dependency Variables

VARIABLE	DESCRIPTION
<code>x_deps</code>	<p>List of source file dependencies corresponding to any segment named <code>x.c</code>, <code>x.asm</code>, <code>x.cpp</code>. Segment must appear in one of the <code>CC_SRC</code>, <code>CPP_SRC</code>, or <code>ASM_SRC</code> lists.</p> <p>For example:</p> <pre>ASM_SRC := ccnucl.asm ccnucl_deps := cicr40.cpy</pre>

## Makefile: Build Override Variables (1)

VARIABLE	DESCRIPTION
CCDEFINES	List of defines to be included in the c89 compiles. -D is <u>not</u> included.
CPPDEFINES	List of defines to be included in the cxx compiles. -D is <u>not</u> included.
CCFLAGS_ <b>x</b>	List of c89 compiler option overrides or additions for module named <b>x</b> . Applied to all CC_SRC listed in makefile for <b>x</b> after defaults.
CCFLAGS_ <b>x_y</b>	List of c89 compiler option overrides or additions for segment named <b>y</b> in module named <b>x</b> . Applied to <b>y</b> after defaults and CCFLAGS_ <b>x</b> .

## Makefile: Build Override Variables (2)

VARIABLE	DESCRIPTION
CPPFLAGS_ <b>x</b>	List of cxx compiler option overrides or additions for program named <b>x</b> . Applied to all CPP_SRC listed in makefile for <b>x</b> after defaults.
CPPFLAGS_ <b>x_y</b>	List of cxx compiler option overrides or additions for segment named <b>y</b> in program named <b>x</b> . Applied to <b>y</b> after defaults and CPPFLAGS_ <b>x</b> .
ASMFLAGS_ <b>x</b>	List of assembler option overrides or additions for program named <b>x</b> . Applied to all ASM_SRC listed in makefile for <b>x</b> after defaults.
ASMFLAGS_ <b>x_y</b>	List of assembler option overrides or additions for segment named <b>y</b> in program named <b>x</b> . Applied to <b>y</b> after defaults and ASMFLAGS_ <b>x</b> .

## Makefile: Build Override Variables (3)

<b>VARIABLE</b>	<b>DESCRIPTION</b>
<b>LDFLAGS_x</b>	List of link option overrides or additions for module name x. Applied to x after default link options for module type (DLL, DLM, LLM).

## Environment File: Variables

VARIABLE	DESCRIPTION
ROOTASMDIRS	List of directories containing assembler source files.
ROOTCDIRS	List of directories containing C source files.
ROOTCPPDIRS	List of directories containing C++ source files
ROOTCPYDIRS	List of directories containing copy files.
ROOTINCDIRS	List of directories containing headers.
ROOMACDIRS	List of directories containing macro files.
ROOTLOADDIRS	Directory containing the executable programs.
ROOTLIBDIRS	Directory containing the definition side decks.
ROOTLSTDIRS	Directory containing the listings.
ROOTOBJDIRS	Directory containing the objects.
ROOTAS_SYSLIB	List of macro/copy PDSs to be used for assemblies.

## Rules File:

- Rules consist of targets, prerequisites, and commands.  
target : prerequisites  
command
- A target is the name of a file to be created or an action to be run (a "phony" rule).
- Prerequisites are a list of input files needed to create the target (optional).
- Commands are actions that make carries out.
- The rules adhere to GNU make syntax.
- The supported targets are listed with the maketpf tool description.

## Configuration File: Build Space Variables

VARIABLE	DESCRIPTION
TPF_ROOT	List of root hfs names, containing TPF source. Roots searched in order specified.
APPL_ROOT	List of root hfs names, containing application (driver) source. Roots searched in order specified.
TPF_BSS_NAME	Basic Subsystem name.
TPF_SS_NAME	Subsystem name. Do not supply unless building a subsystem.
TPF_ROOTPDS	List of PDS names containing TPF macros, copy files, etc. Format specified: ACP.*.RLSE41.
APPL_ROOTPDS	List of PDS names containing application macros, copy files, etc. Format: ACP.*.DRVS

## Configuration File: Build Override Variables

VARIABLE	DESCRIPTION
ASMFLAGS_USER	List of <b>as</b> assembler option overrides or additions. Applied after defaults and makefile overrides.
CCFLAGS_USER	List of <b>c89</b> compiler option overrides or additions. Applied after defaults and makefile overrides.
CPPGLAGS_USER	List of <b>cxx</b> compiler option overrides or additions. Applied after defaults and makefile overrides.
LDFLAGS_USER	List of <b>ld</b> linker option overrides or additions. Applied after defaults and makefile overrides.
TPF_USE_LOCAL_MODS	Setting to include or omit the local_mod directories in the build. Can be YES or NO.

## Control File: Record Structure

- Each record contains the following semicolon delimited fields:
  - ▶ Program name
  - ▶ Number of build passes
  - ▶ Makefile name (with relative pathname)
  - ▶ System allocation (BSS, ALL, EXC)
  - ▶ Object shippable (YES, NO)
  - ▶ Function switches
- Order in file determines build order.
- Comment lines begin with # symbol.

# maketpf

`maketpf pgm [-f] [-q] [target]`

<code>pgm</code>	Defines either the name of the program to build or a makefile name to run. If a program name is specified, the control file is used to find the corresponding makefile. <i>This parameter is required.</i>
<code>-f</code>	Forces all targets to be rebuilt. In effect this is done by hiding all objects in the makefile vpath information.
<code>-q</code>	Turns on quiet mode. By default, standard output is echoed to the terminal.
<code>target</code>	The name of the target to build.

## maketpf Targets (1)

cso	Build the c program and all of its dependencies.
objs	Build all object files for each of the CC_SRC, CPP_SRC, and ASM_SRC lists.
checkenv	Displays a list of environment variables, configuration settings, compile, assemble, and link flags.
clean	Runs clean_lib and clean_obj.
clean_lib	Deletes the executable and related files
clean_obj	Deletes the objects for the CC_SRC, CPP_SRC, ASM_SRC files.
compress	Compress the macro/copy S390 PDS whose name is by MACPDS. This requires the maketpf.compress cmd be available in the PATH on S390.
cpy2pds	FTP all .cpy files found in the first TPF_ROOT directory structure to the S390 PDS whose name is given by MACPDS.

## maketpf Targets (2)

createhfs deletehfs	Creates the output hfs directories for the owning environment, as specified by LIBDIR, OBJDIR, LOADDIR, LSTDIR.
createpds deletepds	Creates the PDS to be used to hold macro and copy files on S390 using name given by MACPDS.
hfs2pds	Runs compress, cpy2pds, and mac2pds.
mac2pds	FTP all .mac files found in the first TPF_ROOT directory structure to the S390 PDS whose name is given by MACPDS.
yyyy.so	Run the rules needed to build the module given by name yyyy.so. All prerequisites will also be built.
xxxx.o	Run the rule needed to build object given by name xxxx.o

## bldtpf

**bldtpf** *cntl.file* [-f] [-q] [-1|-2|-b] [-t BSS|SS|BSSOBJNO]  
[-n nnnn] [-m mmmm] [-l VV] [target]

<i>cntl.file</i>	Defines the name of the control file to process. <i>This parameter is required.</i>
-f	Forces all targets to be rebuilt. Only applies to pass 1.
-q	Turns on quiet mode. By default, standard output is echoed to the terminal.
-1	Run pass 1 only. <ul style="list-style-type: none"> <li>• if cntl file specifies 1 pass needed for link, the target is run with verify link refs.</li> <li>• if cntl file specifies 2 passes needed for link, the target is run without verify link refs.</li> </ul>
-2	Run pass 2 only. <ul style="list-style-type: none"> <li>◆ if cntl file specifies 1 pass needed, that module is not processed.</li> <li>◆ if cntl file specifies 2 passes needed for link, a clean_lib cmd is run first, then the link is run with verify link refs.</li> </ul>
-b	Run both pass 1 and 2, consecutively (default).

## bldtpf (continued)

-t	Type of build to run: <ul style="list-style-type: none"> <li>• BSS - build all SS=BSS,ALL and OBJ=YES,NO (default)</li> <li>• BSSOBJNO - build all SS=BSS,ALL and OBJ=NO</li> <li>• SS - build all SS=EXC,ALL and OBJ=NO</li> </ul>
-n	Specifies the name of the program in the cntl file to begin the build at. Useful for build restarts.
-m	Specifies a pattern. Only lines containing pattern will be built.
-l VV	Specifies generation of a load deck, rather than drive a build. Appends version VV to all loadable names.
target	The name of the maketpf target to build. <ul style="list-style-type: none"> <li>• By default, CSO is the target.</li> <li>• if a target is specified, the build pass is force to 1, as it does not make sense otherwise.</li> </ul>

## maketpf.bsc.convert

maketpf.bsc.convert bscname [-i roothfs] [-o dir]

bscname	Defines the full/relative pathname of the build script to convert. <a href="#">This parameter is required.</a>
-i	Defines the hfs tree to be used to search for the source files corresponding to the build script object name entries.
-o	Defines the directory where the resulting build script is to be written. By default the current working directory is used.

## Single Program Build: Setup

1. Create hfs working directory: `/home/lafer/mywork`
2. Set up the configuration file: `/home/lafer/mywork/maketspf.cfg`
  - a. Set the `TPF_ROOT` and/or `APPL_ROOT` variables. The directory created in step 1 should be first.
  - b. Set the PDS naming convention variables.
  - c. Set the `TPF_BSS_NAME`, `TPF_SS_NAME`, as needed.
3. Create the working PDSs:  
`maketspf createpds`
4. Check out code on change.

## Single Program Build: Sample commands

- If macro or copy files were changed, promote them to PDS:  
`maketpf pgm1 hfs2pds`
- To comple/assemble only the segments that have changed and link:  
`maketpf pgm1`
- To build a specific segment:  
`maketpf pgm1 seg1.o`
- To build all segments belonging to pgm1, regardless of change and link:  
`maketpf -f pgm1`

## Multiple Program Build: Setup

1. Create hfs working directory: `/home/lafer/mywork`
2. Set up the configuration file: `/home/lafer/mywork/maketpf.cfg`
  1. Set the TPF\_ROOT and/or APPL\_ROOT variables. The directory created in step 1 should be first.
  2. Set the PDS naming convention variables.
  3. Set the TPF\_BSS\_NAME, TPF\_SS\_NAME, as needed.
3. Create the working PDSs:  
`maketpf createpds`
4. Check out code on change.
5. Create a control file: `/home/lafer/mywork/my.cntl`
  1. Add records for each program to be built.

## Multiple Program Build: Sample commands

- Build all programs in the control file:  
`bldtpf my.cntl`
- Build all programs in the control file with "base" in the makefile pathname:  
`bldtpf my.cntl -m base`

Any references to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

IBM, MVS, OS/390, and S/390 are trademarks of International Business Machines Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.