



## MakeTPF V2 for TPF41

Copyright International Business Machines Corporation 2004, 2007. All Rights Reserved.

Note to US Government Users Restricted rights - Use, duplication or disclosure restricted by GSA ADP Schedule contract with IBM Corp.

Note: Before using this information and the product it supports, read the general information under "NOTICES" in this document.

## CONTENTS

This file includes the following information:

- 1.0 INTRODUCTION
- 1.1 Change History
- 2.0 PREREQUISITES
- 2.1 HFS Requirements
  - 2.1.1 Required TPF System-Level Directories
  - 2.1.2 Required Customer Application Directories
- 2.2 Data Set Requirements
- 2.3 Required TPF System-Level Data Sets
- 2.4 Required Customer Application Data Sets
- 3.0 INSTALLING MakeTPF for TPF41
- 4.0 CUSTOMIZING MakeTPF for TPF41
  - 4.1 Sample Command Line Build Procedures
  - 4.2 Sample TPF Toolkit Build Procedures
  - 4.3 Additional Information
  - 4.4 Build Script Conversion
  - 4.5 Configuration Files
  - 4.6 Online Help: Manpages
  - 4.7 Creating Application-Specific Environment Files
- 5.0 NOTICES
- 5.1 Trademarks

## 1.0 INTRODUCTION

MakeTPF provides a gnumake-based OS/390 UNIX System Services build solution for TPF 4.1 C and BAL programs. MakeTPF provides tools to:

- Assemble, compile and link source code resident in a hierarchical file system (HFS)
- Produce executable load modules and objects, also stored in the HFS
- Allocate and delete any required directories and partitioned data sets (PDSs)
- Promote macro and copy files between each HFS and PDS
- Promote linked modules and objects between each HFS and PDS.

MakeTPF supports the concatenation of multiple directories in a single build, similar to the way MVS data sets can be concatenated. This eliminates the need to have a complete copy of the source in the directory being used for the build.

MakeTPF for TPF 41 Version 2 is supported by the IBM TPF Toolkit for WebSphere Studio since Interim Fix v2.0.4. This replaces previous support in the Toolkit for MakeTPF for TPF 41 V1. To follow the **Sample TPF Toolkit Build Procedure** outlined in this document you will need to install Interim Fix v2.0.11 or higher. To update TPF Toolkit, click **Help > Software Updates > New Updates** to scan for and apply the latest Interim Fix.

Sample BAL, DLM, DLL, and LLM programs are provided with the MakeTPF tools. Step-by-step directions for building these programs are provided later in this document. For this readme, it is assumed that these programs are installed in the /u/userid/maketpf\_sample directory.

## 1.1 Change History

The following is a summary of changes made to MakeTPF V2 through 09/17/2007. The list includes corrections and enhancements.

- Corrected error with rename of .x files when using a flat hfs structure.
- Corrected error in checkenv Output Directories message.
- Corrected error with processing of .lsc files when using a flat hfs structure.
- Updated assembly step to ignore rc 2.
- Changed line > 80 characters in ASM souce audit, wc -L did not work on USS.
- Corrected inclusion of LMMACRO only when use local mod is YES.
- Corrected looping problem when only one env file was in the maketpf\_list and it did not exist.
- Update maketpf.bsc.convert utility to accept customer application directories and version codes.
- Update tools to eliminate dependency on /bin/ksh symbolic link to /bin/sh.
- Update manpage documentation to remove incorrect information.
- Added support for TPF\_PDS\_DELETE\_CMD, allowing it to be set to bbsrmpds instead of the default tso delete.
- Added support for TPF AR programs with .sqlc and .sqla source files.
- Enhanced the maketpf command to search makefiles for source file names when a source file name rather than a program name is provided as input. Previously maketpf would always create a temporary makefile. Now it will only create a temporary makefile if the owning makefile cannot be found.
- Enhanced the maketpf command to be able to differentiate same-named programs that have different makefile names and system allocation settings in the control file.
- Support has been added for building z/OS offline programs, via EXE targets.
- Support has been added for STLPORT.
- Support for building the CP has been added.
- Corrected error with BAL link rule which forced assembly every time, even when up to date.
- Modified BAL link rule to put BAL objects in OBJ dir and put a link to them in the load directory, so they can be loaded from either place.
- Modified createpds and deletepds rules for ADATA PDS processing error.
- Changed /bin/ksh to /bin/sh in the maketpf\_checkeol and maketpf\_pli scripts.
- Changed ROOTEXPDIR in env\_oco to remove LM layer
- Changed csdsd to appdsd to remove clean\_lib error
- Replaced ADATAPDS with ADATAPDSS in maketpf.env\_TPFUG\_Oct04\_pds
- Removed LMMACROPDS from APPL\_ROOT envs
- Make TPFDF macro PDS conditional on TPF\_SBTPDF function switch
- Eliminated check for OBJ/NOOBJ when building stubs
- Added support for <sys>.function\_switches file
- Backward fit z/TPF MakeTPF Tools APAR PJ30868 to the 41 version to allow the MakeTPF tools to be resolved out of the TPF\_ROOT list and PATH settings. Previously, all of the scripts needed to reside in the same directory.
- Added the USRTPF to control file conversion utility to the package: convert\_usrtpf2cntl.sh
- Added support for z/OS offline program builds.
- Modified the maketpf.bsc.convert command to match the z/TPF copy.
- Incorporated z/TPF MakeTPF enhancements for consistency between 41 and z/TPF toolset.

## 2.0 PREREQUISITES

You need TPF Toolkit Interim Fix v2.0.11 to follow the **Sample TPF Toolkit Build Procedure** outlined in this document.

- To update TPF Toolkit to Interim Fix v2.0.11 or higher, click **Help > Software Updates > New Updates** to scan for and apply the latest Interim Fix.

MakeTPF requires the following on Unix System Services:

- GNU make 3.79.1, available for download from:  
<http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty1.html>  
Once installed, you must verify the gnumake executable (which can be named gnumake, gmake, or make, depending on how the package is installed) is available via the PATH setting on USS so that maketpf can find it.
- The use of the as and ld tools, available from the TPF Build Tools page at:  
<http://www.ibm.com/tpf/download/bldtools.htm>  
Once installed, you must verify the as and ld executables are available via the PATH setting on USS so that maketpf can find them.  
Note: These modules are also shipped with TPF Toolkit.

## 2.1 HFS Requirements

MakeTPF is designed to build source code resident in a hierarchical file system (HFS).

- All source code, including assembler, c, cpp, include, macro, and copy files are expected to reside in the HFS.  
Note that SIP-generated headers also must be copied from the MVS data sets to reside in the HFS if required by any programs. The directory name expected is "sychdr/<sys>" and resides under the TPF41 root. <sys> is a lowercase name corresponding to the value of the TPF\_BSS\_NAME or TPF\_SS\_NAME environment variable, which is explained later in this document.
- MakeTPF has been set up to recognize the TPF41 directory structure as is delivered on the program update tapes (PUTs).
- MakeTPF also is delivered with files (called environment files) that define a sample customer application directory structure. Additional environment files must be created to define each specific customer application directory structure.

### 2.1.1 Required TPF System-Level Directories

Minimally, the TPF header files must be available in an HFS structure in the "include" directory. Again, if needed by any application, the system-generated headers also must reside in the HFS. This means that they must be copied from the ACP.SYMACRO.\*.<SYS> data sets to the sychdr/<sys> HFS. These headers are c\$ldfunc.h, c\$feat.h, and c\$gtsz.h.

### 2.1.2 Required Customer Application Directories

MakeTPF is set up to support any customer application directory structure. The requirement simply is that all directories containing the source code must be identified via the MakeTPF environment files. A sample application directory structure is provided in the maketpf.env\_dlmk file. For this application, the directory structure is as follows (relative to an application directory named /u/userid/drvs), where drvs is a root for a collective set of applications of which dlmk is one group of programs:

/u/userid/drvs/include - common drvs application includes.

/u/userid/drvs/dlmk - dlmk-specific includes (c, cpp, asm, macro, and copy files).

## 2.2 Data Set Requirements

Limitations of the assembler require macro and copy files to still be resident in a partitioned data set (PDS), and limitations of the linker require dlm and library stubs to be resident in a PDS. Link data sets also continue to be used for loading.

The data set naming convention supported by MakeTPF is HLO[.QUAL2].\*[.QUAL3][.SYS]

- The \* symbol is replaced by the MakeTPF solution with CLIB, LINK, MACRO, MACRO.TPFDL, MACRO.TPFMOD, OBJ, SRCE.CP, SRCE.RT, STUB, SYMACRO, SYSRCE

- Dataset name patterns are specified using the TPF\_ROOTPDS variable, specified in the user's build space (the maketpf configuration file.) For example: TPF\_ROOTPDS := ACP.\*.RLSE41
- QUAL2 and QUAL3 can be any valid MVS dataset name qualifier(s). For example: MAKETPF, CUST.APPL, RLSE41, etc. QUAL2 and QUAL3 are optional.
- [.SYS] is a suffix applied by default to subsystem specific datasets. The value of SYS is taken from the TPF\_SS\_NAME, TPF\_BSS\_NAME environment variables, defined in the user's build space. The .SYS suffix is applied to the following datasets: LINK, OBJ, SYMACRO, SYSRCE.
- To omit the use of the [.SYS] suffix from the PDS names, set the following variable in your maketpf configuration file: TPF\_USE\_SYS\_EXT := NO  
This can also be controlled at a system level by updating the default setting of the variable in /u/userid/tools/include41/maketpf.rules\_cfg\_defaults file.

### 2.3 Required TPF System-Level Data Sets

Default names supported: ACP.\*.INTGVP40, ACP.\*.RLSEVP40, ACP.\*.MTPFSRC, ACP.\*.MTPFOUT

Data sets required for the sample build:

- ACP.CLIB.INTGVP40
- ACP.MACRO.INTGVP40
- ACP.SYMACRO.INTGVP40.BSS
- ACP.OBJ.INTGVP40.BSS
- ACP.LINK.INTGVP40.BSS
- ACP.SRCE.RT.INTGVP40
- ACP.STUB.INTGVP40

Optional data sets:

- ACP.MACRO.TPFMOD.INTGVP40 (used if TPF\_USE\_LOCAL\_MODS := YES, by default NO)
- ACP.MACRO.TPFDF.INTGVP40 (used if TPF\_SBTDF := YES is coded, by default NO)

To modify the default TPF system naming convention, see "4.0 CUSTOMIZING MakeTPF for TPF41."

### 2.4 Required Customer Application Data Sets

Default names used in the sample build: USERID.\*.MYPRJ

Data sets required for the sample build:

- USERID.CLIB.MYPRJ
- USERID.MACRO.MYPRJ
- USERID.MACRO.TPFMOD.MYPRJ (used if TPF\_USE\_LOCAL\_MODS := YES)
- USERID.LINK.MYPRJ
- USERID.STUB.MYPRJ
- USERID.ADATA.MYPRJ

## 3.0 INSTALLING MakeTPF for TPF41

1. Unpack MakeTPF41V2.tar.Z:  
cd /u/userid  
pax -rf MakeTPF41V2.tar.Z -ofrom=ISO8859-1,to=IBM-1047

Once unpacked, the directory structure should look as follows:

```
/u/userid/tools
/u/userid/maketpf_sample
```

The MakeTPF tools will require approximately 1.5M when unpacked.

2. Add the MakeTPF41 tools to the PATH:  
PATH=\$PATH:/u/userid/tools

3. The directory where MakeTPF tools were installed must be added to your PATH setting so that the commands can be found and run. This can be done within your \$HOME/.profile or at a system logon profile level. For example, if MakeTPF was unpacked into /u/userid/tools, the PATH would be updated to be:  
PATH=\$PATH:/u/userid/tools
4. The directory where the MakeTPF manpages were installed must be added to your MANPATH setting so that the manpages can be found and run. This can be done in each \$HOME/.profile or at a system logon profile level. For example, if MakeTPF was unpacked into /u/userid/tools, MANPATH would be updated to be:  
MANPATH=\$MANPATH:/u/userid/tools/man
5. If ADATA is to be generated for assembler segments, the directories where the libelf.so and libdwarf.so files reside must be added to the LIBPATH environment variable. These are included with the TPF 41 hfs delivery, in the directories shown, and are also delivered with the TPF Toolkit. For example, if the TPF 41 code is installed under /u/tpf41, LIBPATH would be updated to be:  
LIBPATH=\$LIBPATH:/u/tpf41/debug/tpfdwarf:/u/tpf41/debug/tpfelf

**Note:** For the remainder of this readme file, it is assumed that MakeTPF is installed in /u/userid/tools.

#### **4.0 CUSTOMIZING MakeTPF for TPF41**

1. Edit **/u/userid/tools/include41/maketspf.rules\_env\_pds\_all**
  1. Modify the SYS\_HLQ and APPL\_SYS\_HLQ settings for your site. These define the system-level data set high-level qualifiers.
  2. Modify the SYS\_SUFFIX and APPL\_SYS\_SUFFIX settings for your site. These define the system level QUAL2/QUAL3 values to be recognized by MakeTPF. This enables maketspf to differentiate between system level datasets and user level datasets and enables maketspf to only include system configuration datasets like SYMACRO or SYSRCE for system level datasets and not require them for the user/project level. The user will have MACRO, LINK, OBJ, STUB, and CLIB datasets only. For example, if the user specifies:  
TPF\_ROOTPDS := MYUSERID.\*.MYTPFPRJ  
TPF\_ROOTPDS += ACP.\*.RLSE41  
The full set of datasets listed in the required system level datasets section above will be included in the concatenation lists for the ACP.\* datasets, but only the subset will be included for the MYUSERID.\* datasets.
  3. The system name that is added as the final qualifier on the PDS names (shown as .BSS in the "Required TPF System-Level Data Sets" above) can be turned off by setting TPF\_USE\_SYS\_EXT := NO in the /u/userid/tools/include41/maketspf.rules\_cfg\_defaults file."
2. Edit **/u/userid/tools/include41/maketspf.rules\_cfg\_defaults** and modify the setting of TPF\_DB2LOAD to point to the PDS on your system that contains the DSNHPC program, needed for the SQLC and SQLA conversion support.
3. Edit **/u/userid/tools/include41/maketspf.sqc** and **maketspf.sqa** and update the JCL contained within as needed for your system. This JCL is submitted to perform the SQLC and SQLA conversions. Specifically, job card information and dataset names may need to be updated. The dataset containing the DSNHPC program is controlled by a maketspf variable TPF\_DB2LOAD, set in the previous step.
4. Set up the function switch variables used by MakeTPF to match your system configuration. The function switches are used in the MakeTPF control file to indicate whether or not programs are to be included in the build, based on the value of the function switch. A default function switch of TPF\_SBALL indicates the program is always built. A specific function switch, like TPF\_SBTDF, indicates that a program should only be built, if the value of the function switch is YES. These switches are similar to the FUNC= coding found in the SPPGML macro.

The value of the switches can be set in either of the two ways described below. The values for each of the switches on your system can be obtained from the SIP generated header file named "c\$idfunc.h". For each switch listed in that header, there is a corresponding variable in MakeTPF, with a TPF\_ prefix. For example, if TPDF is enabled on your system, the entry in the header file for the TPDF switch would appear as:

```
"SBTDF",1,
```

In MakeTPF, this must be mapped to a variable as follows (noting the prefix of TPF\_):

```
TPF_SBTDF := YES
```

The supported values for each MakeTPF switch are NO (if 0 in the header) and YES

(if 1 in the header).

To modify the values known to MakeTPF, you can use either of the following methods.

1. Edit `/u/userid/tools/include41/maketpf.rules_functionswitch_defaults` and update the default value of the function switches in that file. For example, modify the `TPF_SBTPDFD` setting to be YES, if you have the TPDFD code installed on your system. This will cause the `MACRO.TPDFD` PDS to be included in the `MACLIB` for assemblies. The default value is NO.
2. Edit `/u/userid/tools/include41_user/<sys>.function_switches` and set the values for each of the function switches in that file.

#### 4.1 Sample Command Line Build Procedures

1. Change to the build directory:
  1. `cd /u/userid/maketpf_sample/drvs/build`
  2. Run all remaining steps from this directory.
  3. By default, the build tools will use the `maketpf.cfg` file in the current directory when building. Therefore you must be in this directory for the `maketpf.cfg` file to be found. If you don't want to run the commands from this directory, optionally, you can export the variable `TPF_CFG` to point to the fully qualified configuration file name. If `TPF_CFG` is set, the configuration file it points to will be used in place of the file named `maketpf.cfg` in the current directory. Also note that `TPF_CFG` can be exported within the file named `.profile` in your home directory, or it can be set on the command line when issuing a `maketpf` command, for example:
 

```
TPF_CFG=/dirname/myfile.cfg maketpf mypgm
```
2. Edit the stub generation makefile to reference the install directory name for the `maketpf.cntl` file:
  1. Edit `/u/userid/maketpf_sample/drvs/build/maketpfstub.mak`
  2. Change the file name specified for `CNTL_SRC` to point to your project level `maketpf.cntl`, created in step 5 above:
 

```
CNTL_SRC := /u/userid/maketpf_sample/drvs/build/maketpf.cntl
```
3. Edit the configuration file to define the build space with directories and data sets to be used in the build:
 

```
/u/userid/maketpf_sample/drvs/build/maketpf.cfg
```

Note that you must leave the `*` symbol as coded:

  1. Set `TPF_ROOT` to the directory containing the TPF source code. This is defaulted to:
 

```
TPF_ROOT := /u/userid/maketpf_sample/tpf41
TPF_ROOT += /u/tpf41/intg
```
  2. Set `TPF_ROOTPDS` to the data set naming convention containing the TPF macro, copy, clib, and stub data sets. This defaulted to:
 

```
TPF_ROOTPDS := ACP.*.RLSEVP40
```
  3. Set `APPL_ROOT` to your working directory and the directory containing the application source code. For example:
 

```
APPL_ROOT := /u/userid/maketpf_sample/drvs
```
  4. Set `APPL_ROOTPDS` to your working PDS naming convention. For this sample build, no system-level data sets are required. For example:
 

```
APPL_ROOTPDS := USERID.*.MYPRJ
```
  5. Set `TPF_BSS_NAME`. By default, this is set to `BSS`. This identifies which basic subsystem PDSs to use in the assemblies, which basic subsystem include directories to use for compiles, and the output HFS directories for any system-specific programs. In this example, the `SYMACRO` PDS name would be: `ACP.SYMACRO.RLSEVP40.BSS`.
  6. Optionally, set the `TPF_SS_NAME`. By default, this is not set. If defined, this identifies which subsystem PDSs to use in the assemblies, which subsystem include directories to use for compiles, and the output HFS directories for any system-specific programs. For example, if `TPF_SS_NAME` was set to `WP`, the two `SYMACRO` PDS names used would be `ACP.SYMACRO.RLSEVP40.WP` and `ACP.SYMACRO.RLSEVP40.BSS`.
  7. Set `USER_VERSION_CODE` to any desired two-character string. This only will be applied to the PDS member names when the programs are copied from the `/load` directory to the `LINK` PDS.
4. Create your application PDSs. This requires one of the program names to be specified as input to `maketpf`:

1. maketpf qpme createpds
2. This should allocate:
  1. USERID.CLIB.MYPRJ
  2. USERID.LINK.MYPRJ
  3. USERID.OBJ.MYPRJ
  4. USERID.MACRO.MYPRJ
  5. USERID.MACRO.TPFMOD.MYPRJ (if  
TPF\_USE\_LOCAL\_MODS := YES)
  6. USERID.STUB.MYPRJ
  7. USERID.ADATA.MYPRJ
5. Create the stubs needed for autocall resolution:
  1. maketpf maketpfstub.mak
  2. This will run a makefile named maketpfstub.mak in the current (build) directory that will read the maketpf.cntl file (located in the maketpf\_sample/drvs//build directory), find any entry that has STUB coded in the STUB column, and generate a DLM-style stub in the USERID.STUB.MYPRJ data set.
6. Build the sample DLL: qpn8
  1. maketpf qpn8
  2. This will create any needed output directories, build all qpn8 segments, link qpn8, and create the qpn8 definition side-deck.
7. Build the sample DLM: qpme
  1. maketpf qpme
  2. This will build all qpme segments and link qpme. No directories should be created because all needed output directories should have been created when qpn8 was built.
  3. This must be built after qpn8 because it requires the qpn8.x definition side-deck for external link resolution.
8. Build the sample LLM: qpn0
  1. maketpf qpn0
  2. This will build all qpn0 segments and link qpn0. No directories should be created because all needed output directories should have been created when qpn8 was built.
9. Build the sample BSO: qpna
  1. maketpf qpna
  2. This will assemble qpna. No directories should be created because all needed output directories should have been created when qpn8 was built.
10. Run a project build using a control file:
  1. bldtpf /u/userid/maketpf\_sample/drvs/build/maketpf.cntl -f
  2. This will build all programs listed in the control file.
  3. The -f option is used to force the rebuild of all objects because they have already been built and the source has not changed.
11. Move all of the programs to the MVS LINK data set:
  1. This can be done per program:
    1. maketpf qpn0 load2pds
    2. maketpf qpme load2pds
    3. maketpf qpn8 load2pds
    4. maketpf qpna load2pds
  2. Or, using the control file:
    1. bldtpf /u/userid/maketpf\_sample/drvs/build/maketpf.cntl load2pds

## 4.2 Sample TPF Toolkit Build Procedures

### Setup

1. Add the MakeTPF41 tools to the PATH in your %TPFSHARE%\bbshtpf.bbs file. For example, if you installed MakeTPF following the instructions in the **Installing MakeTPF for TPF41** section, then you should

- add /u/userid/tools to your PATH environment variable.
2. If ADATA is to be generated for assembler segments, the directories where the libelf.so and libdwarf.so files reside must be added to the LIBPATH environment variable in your %TPFSHARE%\bbshtpf.bbs file. These are included with the TPF 41 hfs delivery, in the directories shown, and are also delivered with the TPF Toolkit. For example, if the TPF 41 code is installed under /u/tpf41, LIBPATH would be updated to be:  
LIBPATH=\$LIBPATH:/u/tpf41/debug/tpfdwarf:/u/tpf41/debug/tpfelf
  3. If you do not already have a connection to the z/OS system where you will be doing the build, create one.
    1. In TPF Toolkit, open the Remote System Explorer (RSE) perspective.  
  
Window > Open Perspective > Other > Remote System Explorer.
    2. In the Remote Systems view, expand the **New Connection** node.
    3. Expand the **z/OS** node.
    4. Specify the hostname for the remote system.
    5. Click **Finish**.
    6. In the Remote Systems view, expand the connection you have just created to reveal the **USS Files** subsystem.
    7. Expand the **USS Files** subsystem to reveal the **Root** and **Home** filters. Change the properties of the filters to point to the appropriate location on your host file system. For example, the **Home** filter should point to your user directory (/u/userid).
    8. Expand the **Home** directory to reveal the contents of your user directory.

### Build the sample programs

1. Create a TPF Project to work with the sample.
  1. Open the TPF Toolkit perspective, if it is not already open.  
Window > Open Perspective > TPF Toolkit.
  2. Right click inside the TPF Project Navigator view and select New > Project
  3. In the New Project wizard dialog, select TPF in the left panel and Project in the right panel.
  4. Click **Next** to proceed to the New TPF Project page.
  5. Specify the project name: **maketpf**.
  6. Browse for the Remote Working directory.  
Select /u/userid/maketpf\_sample/drvs/build. By default, the build tools will use the maketpf.cfg file in this directory.
  7. Click **Next** to proceed to the New TPF Project Filter page.
  8. Specify the name of the TPF filter that you will use to group your sample application files. For example: **Sample Files**.
  9. Click **Next** to proceed to the New Filter String page.
  10. Click **Add** to browse for your project the sample application directory. For example, /u/userid/maketpf\_sample.
  11. Click Finish to create the TPF project. In the TPF Project Navigator view, you should see all the relevant files when you expand the Sample Files filter under the maketpf project.
2. Edit the configuration file to define the build space with directories and data sets to be used in the build.
  1. Right click the **maketpf** project and select Properties.
  2. Select **TPF Make Configuration** in the left navigation panel.
  3. In the **TPF System** tab, the **Basic sub-system** name defaults to BSS. Specify the appropriate value for your environment. This identifies which basic subsystem PDSs to use in the assemblies, which basic subsystem include directories to use for compiles, and the output HFS directories for any system-specific programs. In this example, the SYMACRO PDS name would be:  
ACP.SYMACRO.INTGVP40.BSS.
  4. Optionally, set the **Sub-system Name**. By default, this is not set. If defined, this identifies which subsystem PDSs to use in the assemblies, which subsystem include directories to use for compiles, and the output HFS directories for any system-specific programs. For example, if TPF\_SS\_NAME was set to WP, the two SYMACRO PDS names used would be ACP.SYMACRO.INTGVP40.WP and

ACP.SYMACRO.INTGVP40.BSS

5. In the **TPF source** list, add the directories containing the TPF source code. By default, /u/userid/maketpf\_sample/tpf41 and /u/tpf41/intg are listed. Replace them with the appropriate paths. For example, replace userid with your userid. Use the **Up** and **Down** button to order the items in the list.
  6. In the **Macro and copy file PDS** list, add the data set naming convention containing the TPF macro, copy, clib, and stub data sets. This defaulted to: ACP.\*.RLSEVP40
  7. Click the **Application** tab to specify application specific values.
  8. In the **Source** list, add your working directory and the directory containing the application source code. By default, /u/userid/maketpf\_sample/drvs is specified. Replace it with the appropriate paths. For example, replace userid with your userid. Use the **Up** and **Down** button to order the items in the list.
  9. In the **Macro and copy file PDS** list, specify your working PDS naming convention. By default, USERID.\*.MYPRJ is specified. Remove the incorrect entries and add the appropriate ones. For example, replace USERID with your userid. Use the **Up** and **Down** button to order the items in the list. For this sample build, no system-level data sets are required.
  10. Optionally, click the **Advanced** tab and set USER\_VERSION\_CODE to any desired two-character string. This only will be applied to the PDS member names when the programs are copied from the /load directory to the LINK PDS.
3. Edit the stub generation makefile so that it references the maketpf.cntl file in your project working directory:
    1. Expand the **Sample Files** filter under your project.
    2. Expand the drvs folder.
    3. Expand the build folder.
    4. Double click the file maketpfstub.mak
    5. Change the file name specified for CNTL\_SRC to point to your project level maketpf.cntl. For example, replace userid with your userid.
  4. Create your application PDSs.
    1. Expand the **Sample Files** filter under your project.
    2. Expand the drvs folder.
    3. Expand the dlmk folder.
    4. Right click on qpme.mak and select **MakeTPF > Allocate Output PDS (createpds)** from the pop-up menu.
    5. This following PDS should be allocated:
      1. USERID.CLIB.MYPRJ
      2. USERID.LINK.MYPRJ
      3. USERID.OBJ.MYPRJ
      4. USERID.MACRO.MYPRJ
      5. USERID.STUB.MYPRJ
  5. Create the stubs needed for autocall resolution:
    1. Locate maketpfstub.mak under the build folder.
    2. Right click on maketpfstub.mak and select **MakeTPF > Compile Only (objs)** from the pop-up menu. This will run the maketpfstub.mak makefile in the project build directory. It will read the maketpf.cntl file (located in the same directory), find any entry that has STUB coded in the STUB column, and generate a DLM-style stub in the USERID.STUB.MYPRJ data set.
  6. Build the sample DLL: qpn8
    1. In the TPF Project Navigator, find qpn8.mak. It should be located under drvs/dlmk directory.
    2. Select qpn8.mak, right click and select **MakeTPF > Build (Compile and Link)** from the pop-up menu.
    3. This will create any needed output directories, build all qpn8 segments, link qpn8, and create the qpn8 definition side-deck.
    4. You should find QPN8NN in the dlmk/load directory, where NN is the version code.
  7. Build the sample DLM: qpme

1. In the TPF Project Navigator, find qpme.mak. It should be located under drvs/dlmc directory.
  2. Select qpme.mak, right click and select **MakeTPF > Build (Compile and Link)** from the pop-up menu.
  3. This will build all qpme segments and link qpme. No new directories should be created because all needed output directories should have been created when qpn8 was built.
  4. This must be built after qpn8 because it requires the qpn8.x definition side-deck for external link resolution.
  5. You should find QPMENN in the dlmc/load directory, where NN is the version code.
8. Build the sample LLM: qpn0
1. In the TPF Project Navigator, find qpn0.mak. It should be located under drvs/dlmc directory.
  2. Select qpn0.mak, right click and select **MakeTPF > Build (Compile and Link)** from the pop-up menu.
  3. This will build all qpn0 segments and link qpn0. No new directories should be created because all needed output directories should have been created when qpn8 was built.
  4. You should find QPN0NN in the dlmc/load directory, where NN is the version code.
9. Build the sample BSO: qpna
1. In the TPF Project Navigator, find qpna.mak. It should be located under drvs/dlmc directory.
  2. Select qpna.mak, right click and select **MakeTPF > Build (Compile and Link)** from the pop-up menu.
  3. This will assemble qpna. No directories should be created because all needed output directories should have been created when qpn8 was built.
  4. You should find QPNANN in the dlmc/load directory, where NN is the version code.
10. Instead of building each program one by one, you can also build the project using a control file. Building the project using a control file will build all the programs listed in the control file.
1. Right click the maketpf project in the TPF Project Navigator and select **MakeTPF > Build with Force (Compile and Link)** from the pop-up menu.  
Note: The force option is used to force the rebuild of all objects because they have already been built and the source has not changed.
  2. This will invoke bldtpf to build all programs listed in the control file and generate all the modules in dlmc/load directory.
11. You are now ready to create a loadset to load these programs to your TPF test system to run.

### 4.3 Additional Information

maketpf41, bldtpf41:

- For forward compatibility, a maketpf41 and bldtpf41 command is also delivered. These commands force the MAKETPF\_VERSION to 41, and then pass all arguments to the maketpf or bldtpf scripts. Currently, only MAKETPF\_VERSION 41 is supported, so maketpf41 and maketpf can be used interchangeably.

TPF definition side-decks (DSDs): TPF DSDs must reside in the HFS

- MakeTPF expects that DSDs reside in the HFS. For this sample build, the only DSD needed was built as part of the example. If TPF DSDs are needed for link resolution, they must be moved (in binary) from the MVS data set to the HFS in a directory named <TPF\_ROOT>/export.
- Building with circular external link references. PGM1 requires a DSD from PGM2, and PGM2 needs a DSD from PGM1 for both to link cleanly. This is resolved as follows:
- In maketpf.cfg, set TPF\_VERIFY\_LINK\_REFS := NO

- Build PGM1. The maketpf program will assemble/compile/link PGM1, creating pgm1.x, even though PGM1 itself is not executable.
- Edit maketpf.cfg and set TPF\_VERIFY\_LINK\_REFS := YES
- Build PGM2. The maketpf program will link pgm2 cleanly and produce pgm2.x.
- Link PGM1 again. The maketpf program will re-link pgm1 with pgm2.x.
- Note that within the control file, PGM1 should be listed first with two build passes. PGM2 should be listed second, needing only one build pass. When processed by bldtpf, the manual scenario above will be followed, where PGM1 is linked twice; once without and then again with PGM2.x, and PGM2.x is linked only once.

#### 4.4 Build Script Conversion

- The example build was delivered with the makefiles already created; however, a utility named "maketpf.bsc.convert" also is provided to convert build scripts to MakeTPF-format makefiles. You can run conversion tool either from the command line or from TPF Toolkit.
- To convert the sample build scripts from the command line:
  - mkdir -p bscconvert/drvs/dlmk
  - cd /u/userid/bscconvert/drvs/dlmk
  - maketpf.bsc.convert -i /u/userid/maketpf\_sample/drvs /u/userid/maketpf\_sample/drvs/dlmk/qpmeps.bsc
- This will convert qpmeps.bsc to qpme.mak and write it in the current directory.
- To convert the sample build scripts using TPF Toolkit
  - Expand the **Sample Files** filter under your **maketpf** project in TPF Toolkit
  - Expand the appropriate folders to locate qpmeps.bsc.
  - Right click on qpmeps.bsc and select **Convert to makefiles** from the pop-up menu.
  - The Build Script to Makefile dialog is displayed with the appropriate values filled in. By default, the resulting makefile will be generated in the same directory as the selected build script file. Make any necessary modifications and click **OK**.
- This will convert qpmeps.bsc to qpme.mak in the output location that you specified in the dialog.

#### 4.5 Configuration Files

- By default, a file named maketpf.cfg is used as the configuration file name.
- Also by default, the configuration file is expected to reside in the directory that maketpf is running from.
- To change this behavior, set environment variable TPF\_CFG, and its value will be used by maketpf and bldtpf to determine the configuration file location.
- export TPF\_CFG=/mydir/my.cfg
- TPF\_CFG=/mydir/maketpf.cfg maketpf

#### 4.6 Online Help: Manpages

- Reference information for the tools and associated files is provided in manpage format and can be viewed using the "man" command. Examples:
  - man maketpf  
Displays the maketpf command syntax and lists the supported targets/rules.
  - man maketpf.cfg  
Displays the environment variables that can be set in the configuration file.

#### 4.7 Creating Application-Specific Environment Files

- New environment files can be set up for each unique customer application HFS.
- The environment files must be named maketpf.env\_<appl>, where <appl> is any unique string.
- If there are any associated PDS requirements for the environment, the PDS environment file must be named maketpf.env\_<appl>\_pds
- All environment variables shown in the maketpf.env\_dlmk and maketpf.env\_dlmk\_pds must be set in all new environment files. The

variables must be set to the directory name or names containing the associated source or output file types. Each variable can point to the same directory for a flat HFS structure, or be as specific to point to unique directories for each file type depending on your HFS setup. Typically, output variables list only one directory path. In an instance where more than one directory path is listed, the first one listed is used as the output directory.

- To use the environment in a makefile, set:  
`maketpf_env := <appl>`  
 To make the <appl> environment the *owning* environment. Additional environments can be added for source file resolution, such as:  
`maketpf += base_rt`
- If <appl> is listed first in the maketpf\_env list, it is considered the owning environment and any output file (object, load module, and so on) will be written to the output directories defined by env\_<appl>.
- If <appl> is listed other than first in the maketpf\_env list, it is used for locating source code (headers, .c, .x, and so on) only. No objects are taken from environments that are listed other than first.

## **5.0 NOTICES**

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
 IBM Corporation  
 North Castle Drive  
 Armonk, NY 10504-1785  
 U.S.A

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
 Department 830A  
 Mail Drop P131  
 2455 South Road  
 Poughkeepsie, NY 12601-5400  
 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee. Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

## **5.1 Trademarks**

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM  
 MVS  
 OS/390

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

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