IBM Software Group

z/TPF Build Education

Taking the mysteries out of the z/TPF build process

TPF Ongoing Education

Brian Laferriere

October 2004

AIM Core and Enterprise Solutions

IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

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.

Topics

- Build
  - Overview
  - Build Environment
  - Directory Structure
  - Program Types and Formats
  - Datasets
  - JCL
- MakeTPF Build Solution
  - Overview
  - Advantages
  - Tools
  - Files
- System Generation Procedures

Topics

- Sample Customer Application
- What you can do today to prepare for z/TPF
  - Define your directory structure
  - Install MakeTPF for 4.1
  - Create makefiles from build scripts

Overview: What is build?

- z/TPF system generation
  - System initialization (SIP)
    - Definition of the z/TPF system configuration:
      - Database layout and memory configuration
      - Loosely coupled vs. single processor
      - Number of subsystems and their names
      - SIP deck assembly deck (formerly Stage 1 Deck)
      - Face Table generation
  - Program assemble, compile, and link
    - Online programs (CP, CIMR, Keypoint, Real-time)
    - Offline programs
- Customer program assemble, compile, and link
  - System programs
  - Application programs

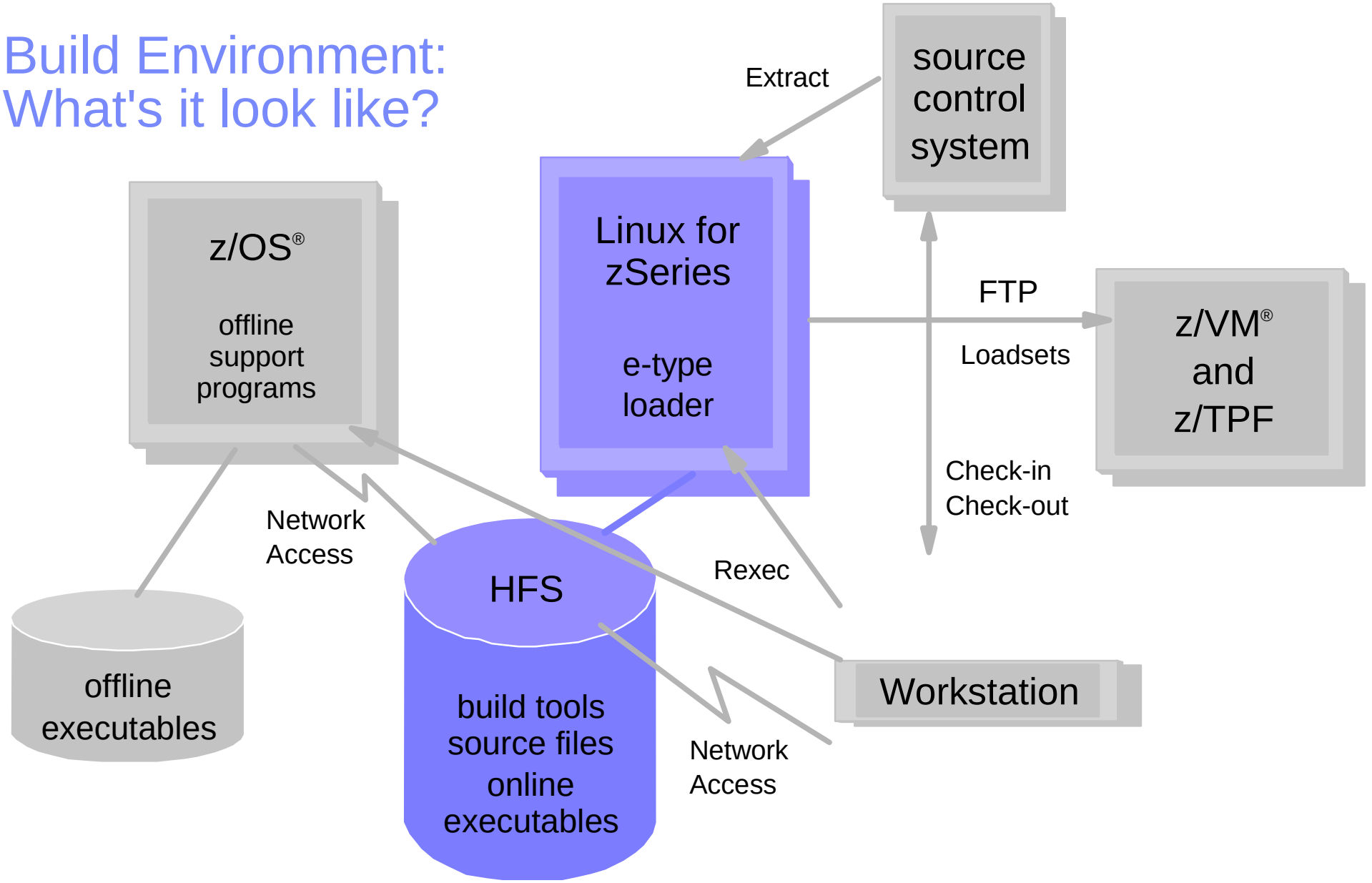
Overview: What has changed?

- New build platform
  - Builds run primarily on Linux
  - Unix System Services on z/OS® used for remaining z/OS based offline programs
- New tools and languages
  - MakeTPF Build Tools replace SIP stage 1 & 2
  - GNU Make and Korn Shell replace JCL
  - GNU Compiler Collection (GCC) replaces z/OS C/C++
- New file names
  - Directories replace datasets and pathnames are required
  - Source Code Managers (SCM) must handle long file names
- New program formats
  - Elf shared objects replace load modules and BAL objects

Overview: What has improved?

- Consistency
  - All programs are now built with the same tools
  - System and development builds use the same tools
- Integration
  - MakeTPF Build Tools are integrated with the TPF Toolkit for Websphere® Studio
- Time
  - A full system build now runs in 2 1/2 hours
  - A comparable TPF 4.1 build requires 6-8 hours

Build Environment: What's it look like?



Build Environment: Linux for zSeries

- What happens here?
  - Building
    - Linux utility program builds
    - SIP deck assembly
    - Face table generation
    - Assemble, compile, link of all online programs
    - Assemble, compile, link of all Linux offline programs
  - Loading
    - OLDR loads (via FTP) to TPF
  - Access to source code
    - Copy of the TPF source code from the SCM
    - Check-in/out capability to the SCM (optional)
  - Sharing of source code and binaries
    - Serve mounts to z/OS, Windows
  - Editing of source code

Build Environment: Linux for zSeries

- Requirements
  - Linux for zSeries (64-bit with 32-bit compatibility mode)
  - IBM z/TPF Database Facility (z/TPFDF) Enterprise Edition Version 1 Release 1
  - GNU compiler collection (GCC) components
    - gcc-3.4.1 or later (built in cross-compiler mode specifically for z/TPF)
    - g++-3.4.1 or later (build in cross-compiler mode specifically for z/TPF)
    - binutils-2.15 or later
  - HLASM
    - Statement of Direction: Provide HLASM capability on Linux for zSeries prior to, or at the time of, the general availability
  - glibc and libstdc++ runtime libraries modified for z/TPF
  - Korn Shell: pdksh
    - Korn shell is used over bash for performance reasons

Build Environment: z/OS with USS

- What happens here?
  - Assemble, compile, link of all z/OS offline programs
  - LGF, GDS, TPAPE/VTAPE, VRDR loads to TPF
  - Access to code, via mounts to Linux directories
  - Editing of source code
- Requirements
  - IBM z/OS Version 1 Release 3, or later release, including High Level Assembler (HLASM) Release 5
  - IBM DATABASE 2® (DB/2) Server for OS/390 Version 5, or later release
  - IBM Enterprise PL/I for z/OS Version 3 Release 3, or later release
  - GNU Make (3.79.1)
  - Korn Shell (pdksh)
  - NFS/SMB Client Capability

Build Environment: z/OS with USS

- Remaining z/OS offline programs
  - amx1, aslo1, bgml, bpr0, brfa, cbq4, cbq5, cbq6
  - cbqprt, chqi, dataread, db2pp, dfad, fmtr, hlst, ostg
  - ppcp, sadprt, stc, iptprt, tpfldr

Build Environment: SCM

- What happens here?
  - This is the source code repository
- Requirements
  - Must support long file names (greater than 8 characters)
  - Must support file extensions (.asm, .c, .cpp, etc.)
  - Must support pathnames (base/macro/sip/gensip.mac)
- Considerations
  - Linux command line interface
  - Windows command line interface (if you plan to use the IBM TPF Toolkit for WebSphere Studio V2, this is required)
  - Version control
  - Concurrent development support

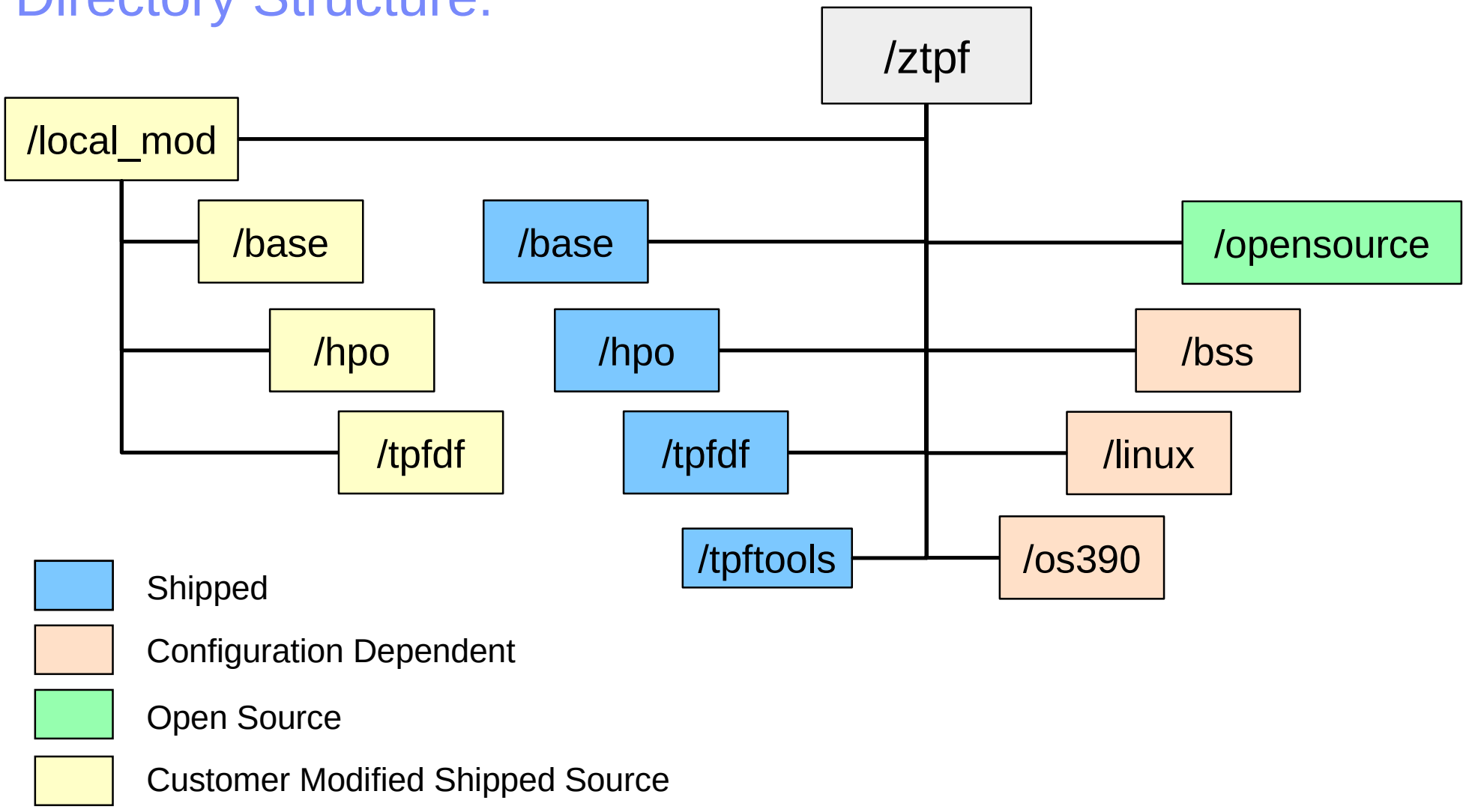
Build Environment: Windows

- What happens here?
  - This is the developer's workstation
- Considerations
  - IBM TPF Toolkit for WebSphere Studio V2
  - SCM Client and/or command line interface
  - Telnet capabilities to Linux and z/OS
  - Access to Linux source code directories
    - SMB
    - NFS
    - TPF Toolkit RSE

Directory Structure: Organization

- z/TPF shipped product files
- Files that must be built by the customer
  - System configuration dependent source files and programs
  - Operating system dependent (offline and utility) programs
- Files that must be obtained and build by the customer
  - Opensource files, like
    - glibc
    - libstdc++ libraries
- Customer modifications to z/TPF shipped files

Directory Structure:



Directory Structure: Shipped Product Files

- Named by product, feature
  - base/
  - hpo/
  - tpfdf/
  - tpftools/
    - Note: The MPIF and AR features are now included in the BASE product.
- Contains
  - All shippable source
  - Binaries (elf objects and shared objects) and listings for configuration-independent source files and programs
- Are shipped as relative directory names
  - You choose the owning root directory name, for example: /ztpf

Directory Structure: base/

- ascl/
  - lib
  - load
  - lst
  - obj
- cb2b/
  - lib
  - load
  - lst
  - obj
- cdec/
  - lib
  - load
  - lst
  - obj
  - src
- cimr/
- cntl/
- cp/
- debug/
  - tpfdwarf/
    - cmplrs/
  - tpfelf/
- exp/
- filesys/
  - exp/
  - ffs/
  - include/
    - sys/
      - ffs/
      - mfs/
      - pfs/
      - tfs/
    - tpf/
  - lib/
  - load/
- lst/
- mfs/
- obj/
- pfs/
- tfs/
- tpf/
- vfs/
- include/
  - ascl/
  - dce/
  - isc/
  - sys/
  - sip/
    - globals/
      - tpf/
  - tpf/
- kpt/
- lib/
- load/
- lst/
- macro/
  - sip
    - globals
- obj/
- oco/
  - lib/
  - load/
  - stdlib/
  - stdload/
- ol/
- openssl/
  - ...
- rt/
  - ar
- samples/
  - glinit
  - jcl
  - sip
- stdlib/
- stdload/
- tpf_mail/
  - ...
- util/
  - include/
  - migrationtools/
  - src/
    - calst/
    - fctbg/
  - tools/
- xml4c/
  - ...

Directory Structure: hpo/

- cp/
- lib/
- load/
- lst/
- obj/
- rt/

Directory Structure: tpfdf/

- exp/
- include/
- lst/
- macro/
  - spm/
- obj/
- rt/

Directory Structure: tpftools/

- include_ztpf/
- include_ztpf_user/
- man/
  - man1/
  - man3/
  - man5/
- samples/

Directory Structure: System Configuration Dependent Files

- Named for corresponding TPF system
  - bas/ - base only system (SSDEF not coded in SIP deck)
  - bss/ - basic subsystem (SSDEF BSSGEN=YES)
  - nnnn/ - basic subsystem (SSDEF BSSGEN=YES,SSNAME=nnnn)
  - ssss/ - subsystem (SSDEF BSSGEN=NO,SSNAME=ssss)
- Contains
  - All system configuration dependent source files, including the SIP Deck and the SIP generated macros source files
  - Binaries (elf objects and shared objects) and listings for configuration source files and programs
- Are generated relative to the chosen root directory

Directory Structure: bss/

- include/
  - tpf/
- jcl/
- lib/
- load/
- lst/
- macro/
- obj/
- src/
- stdlib/
- stdload/
- user/
  - include/
    - tpf/
  - macro/

Directory Structure: Operating System Dependent Files

- Named for the target operating system
  - linux/ - Linux offline and utility programs
  - os390/ - z/OS offline programs
- Contains
  - Binaries (elf objects and shared objects) and listings for off-line, operating system specific source files and programs
  - Note that the source for the offline programs is contained under the base/ol and base/util directories.
- Are generated relative to the chosen root directory

Directory Structure: linux/

- bin/
- lib/
- lst/
  - calst/
- obj/
  - calst/

Directory Structure: os390/

- bin/
- lst/
- obj/

Directory Structure: Opensource Files

- Named for corresponding opensource product or the files produced during their build
- Contains
  - Customer obtained opensource files
  - Binaries (elf objects and shared objects) and listings for the opensource files and programs
- Notes
  - MakeTPF format makefiles are provided in the base/ directory structure for the open source programs required for the build, to ensure a consistent build process
- Must be created relative to the chosen root directory
  - /ztpf/opensource

Directory Structure: opensource/

- apache/
- glibc/
- include/
- lib/
- libstdc++/
- load/
- locale_gen/
- stdlib/
- stdload/

Directory Structure: Customer Modified Shipped Files

- Named local_mod/
  - This directory is optional
- Intended use
  - To house customer modifications to z/TPF shipped source files
    - For example, changes to user exits, usr.cntl
    - The full relative pathname and filename of the updated file must be specified. For example, local_mod/base/cp/ccnucl.asm
  - To simplify a directory based 3-way merge of customer modified source with future z/TPF updates
  - Does not apply to system configuration dependent source
- Must be created relative to the chosen root directory
  - /ztpf/local_mods

Directory Structure: local_mod/

- base
  - ...
- hpo
  - ...
- tpfdf
  - ...

Program Types: TPF

Description	MakeTPF Target	Format
C and BAL Real Time application programs	APP	elf shared object (tpfgcc)
Archives - collection of elf object files	ARCHIVE	elf shared object (tpfgcc)
z/TPF CIMR components ACPL, CPS0, ICDF, IPAT, IPLB, RIAT, SIGT, USR1, USR2	CIMR	elf shared object (tpfgcc)
Face Table	FCTB	elf shared object (tpfgcc)
Keypoints (including general file keypoints)	KPT	elf shared object (tpfgcc)
IPLA	IPL	object module

Program Types: Off-line

Description	MakeTPF Target	Format
Archive - collection of elf object files	ARCHIVE	elf shared object (gcc)
Linux offline and utility programs	EXE	elf shared object (gcc)
z/OS offline programs	EXE	load module
Standalone dump processor	SADUMP	object module

Datasets

- One partitioned dataset remains for z/OS offline program load modules
  - Offline program object modules stored in the hfs on USS
- Naming convention: <hlq>.LINK.<version>.<ssname>
- For example: ZTPF.LINK.REL11.BSS
- All other datasets eliminated or replaced with temporary datasets

JCL

- Any remaining JCL is used for running z/OS offline programs only
- JCL is no longer used for build purposes

MakeTPF Build Solution: Overview

- Is a complete set of tools used for assembling, compiling, and linking both TPF system and customer application programs
- Is scalable, enabling builds of a single program, a set of programs (project), or a full system
- Is implemented using Korn shell scripts and GNU Make
- Is available on both Linux and OS/390 Unix platforms, for both online and offline program builds

MakeTPF Build Solution: Advantages

- Single build interface
  - Provides centralized assemble, compile, and link rules
  - Defines default assemble, compile, and link options
  - Can be used for both full system and development builds
  - Is integrated with the TPF Toolkit For Websphere Studio
  - Supports circular external references with an optimized two-pass build
- Is highly customizable
  - Can support unique customer application directory structures
  - Can support unique customer rules and build audits
- Defines each program's build information within the makefile
  - Program name, program type, associated source files, dependencies
  - Compile, assemble, and link option overrides

MakeTPF Build Solution: Advantages

- Provides an directory-based solution for customer modifications to TPF source code
  - Simplifies the maintenance process
- Can support multiple TPF configurations under one root
  - Supports TPF configuration-dependent source and programs
  - Enables the use of the same makefile with all configurations
  - Optimizes the build of configuration-independent programs, for multi-subsystem builds
- Supports concatenation of multiple source code hfs's within a single build
- Minimizes makefile knowledge required

MakeTPF Build Solution: Tools & Files

- Tools
  - maketpf
  - bldtpf
  - loadtpf
- Files
  - Makefile
  - Configuration File
  - Environment File
  - Control file
  - Rules File

MakeTPF Build Solution: maketpf

- A single program builder for MakeTPF format makefiles
  - Used to assemble, compile, and link an individual TPF system or customer application program
- Can be used to assemble, compile, and link programs by:
  - Program name, where the makefile information is derived from the control file
  - Makefile path and file name
- Can be used to assemble or compile individual source files
- Requires a configuration file to define the build space

MakeTPF Build Solution: maketpf

- Usage
  - `maketpf pgm|mak|seg [-f|-f1] [-q] [target ...]`
- Examples
  - Assemble source and link only the updated source:
    - `maketpf ciso`
  - Assemble all source and link, regardless of change:
    - `maketpf -f ciso`
  - Assemble a source file relative to its owning executable:
    - `maketpf cps0 ccnucl.o`
  - Assemble a source file, independent of a makefile:
    - `maketpf ccnucl.asm`
  - Assemble all source and link, regardless of change, given a makefile:
    - `maketpf -f /ztpf/base/cp/cps0.mak`

MakeTPF Build Solution: bldtpf

- A multiple program builder for MakeTPF format makefiles
  - Used to assemble, compile, and link a set of TPF system or customer application programs listed in a specified control file
- Also used to drive
  - SIP deck assembly
  - Face table generation
  - Load deck generation
  - Stub library source generation
  - PAT-to-control file conversions
- Requires a configuration file to define the build space

MakeTPF Build Solution: bldtpf

- Usage
  - `bldtpf -tpf [-1|-2-|-b] [-f|-f1] [-j jobs] [-k] [-q] [-m mmmm] [-n nnnn] [-noobj] maketpf.cntl [target...]`
  - `bldtpf -ol [-1|-2-|-b] [-f|-f1] [-j jobs] [-k] [-q] [-m mmmm] [-n nnnn] [-noobj] maketpf.cntl [target...]`
  - `bldtpf -util [-1|-2-|-b] [-f|-f1] [-j jobs] [-k] [-q] [-m mmmm] [-n nnnn] [-noobj] maketpf.cntl [target...]`
  - `bldtpf -tld [-v VV] maketpf.cntl`
  - `bldtpf -ald [-v VV] maketpf.cntl`
  - `bldtpf -fctb sip.asm`
  - `bldtpf -fctb_src sip.asm`
  - `bldtpf -ipat maketpf.cntl`
  - `bldtpf -sip sip.asm`
  - `bldtpf -stub maketpf.cntl`
  - `bldtpf -pat2ctl patfile`

MakeTPF Build Solution: bldtpf

- Examples
  - Assemble, compile, and link all TPF online programs:
    - `bldtpf -tpf -b -f /ztpf/base/cntl/tpf.cntl`
  - Assemble, compile, and link all filesystem programs:
    - `bldtpf -tpf -2 -f -m filesys/ /ztpf/base/cntl/tpf.cntl`
  - Assemble the SIP deck:
    - `bldtpf -sip /ztpf/bss/src/sip.asm`

MakeTPF Build Solution: Makefile

- Defines
  - The name and type of the program
  - The base names of the source files required to build the program
    - Note that base names must therefore be unique across the directories for the file type, defined in the set of environments referenced in the makefile
  - Where the source files reside, via environments
  - Where the output files are to be created, also via environments
  - Assemble, compile, or link option overrides, specific to the program or any of its source files
  - Dependency relationships to include, macro, or copy files (optional)
- Uses GNU Make syntax to set MakeTPF-defined schema variables
- Includes a common maketpf.rules file

MakeTPF Build Solution: Makefile

- Example
  - Filename:
    `base/rt/cfca.mak`
  - Content:

```
APP := CFCA
APP_ENTRY := CFCA

maketpf_env := base_rt
maketpf_env += system

C_SRC := cfca.c

include maketpf.rules
```

MakeTPF Build Solution: Configuration file

- Defines the working build space
  - The concatenation list of directory structures to be included in the build
    - The order specified defines search order
    - The first root directory specified is used for the output file creation
  - The system and or subsystem to build for
    - Identifies which directory of system configuration dependent source to use
  - Build control options
    - For example, whether or not to keep listings
  - Temporary assemble, compile, and link option overrides

MakeTPF Build Solution: Configuration file

- Example
  - Filename:
    `mywork/build/maketpf.cfg`
  - Content:

```
TPF_ROOT := /home/lafer/mywork
TPF_ROOT += /ztpf

TPF_BSS_NAME := BSS
TPF_SS_NAME := SS1

CFLAGS_USER := -O0
```

MakeTPF Build Solution: Environment file

- Defines the location of source and output files for logical groupings of programs
  - Maps each directory within the structure to a fixed set of MakeTPF schema variables, based on the directory content
  - Isolates the makefiles from the directory structure, thereby allowing the directory structures to evolve without having to update individual makefiles
- Can be created an customized for customer application directory structures
  - Will support both flat and multilevel deep directory structures
- Directories can be referenced in multiple environment files
  - Allows for common directories to be shared across environments

MakeTPF Build Solution: Environment file

- Example
  - Filename:
    `tpftools/include_ztpf/maketpf.env_base_rt`
  - Content:

```
ROOTLIBDIRS := $(foreach d,$(TPF_ROOT),$d/base/lib)
ROOTLOADDIRS := $(foreach d,$(TPF_ROOT),$d/base/load)
ROOTEXPDIRS := $(foreach d,$(TPF_ROOT),$d/base/exp)
ROOTOBJDIRS := $(foreach d,$(TPF_ROOT),$d/base/obj)
ROOTLSTDIRS := $(foreach d,$(TPF_ROOT),$d/base/lst)
ROOTINCDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/include)
ROOTINCDIRS += $(foreach d,$(TPF_ROOT_LM),$d/base/../opensource/include)
ROOTINCDIRS += $(foreach d,$(TPF_ROOT_LM),$d/base/../opensource/include/g++)
ROOTINCDIRS += $(foreach d,$(TPF_ROOT_LM),$d/base/../opensource/include/g++/backward)
ROOTMACDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/macro)
ROOTMACDIRS += $(foreach d,$(TPF_ROOT_LM),$d/tpfdf/macro/spm)
ROOTCPYDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/rt)
ROOTCDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/rt)
ROOTCDIRS += $(foreach d,$(TPF_ROOT_LM),$d/base/rt/ar)
ROOTCPPDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/rt)
ROOTCNTLDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/cntl)
ROOTTOOLDIRS := $(foreach d,$(TPF_ROOT_LM),$d/base/util/tools)
```

MakeTPF Build Solution: Rules file

- Defines
  - The single source for the assemble, compile, and link rules
  - The default assemble, compile, and link options
- Resolves full source file path names, using:
  - The source file base name provided in the makefile
  - The environments listed in the makefile
  - The TPF_ROOT or APPL_ROOT directory concatenation lists

MakeTPF Build Solution: Control Files

- Central point of information regarding all programs
  - Replace ibmpal.cpy, usrtpf.cpy, and sppgml.mac
- Defines build order
  - Program built in order listed
- Spreadsheet like format (semicolon delimited columns of data)
- TPF System and Customer application program control file
  - base/cntl/tpf.cntl
  - base/cntl/usr.cntl
- Can be set up for project builds

MakeTPF Build Solution: Control Files - Shipped

- base/cntl/tpf.cntl
  - base/cntl/tpf_util_linux.cntl
  - base/cntl/tpf_stdlib.cntl
  - base/cntl/tpf_cimr.cntl
  - base/cntl/tpf_kpt.cntl
  - base/cntl/tpf_app_base_ux.cntl
  - base/cntl/tpf_app_base.cntl
  - base/cntl/tpf_app_hpo.cntl
  - base/cntl/tpf_app_tpfdf_ux.cntl
  - base/cntl/tpf_app_tpfdf.cntl
  - base/cntl/tpf_app_base_xv.cntl
  - base/cntl/tpf_app_tpfdf_xv.cntl
  - base/cntl/tpf_ol_linux.cntl
  - base/cntl/tpf_ol_os390.cntl
  - base/cntl/tpf_app_usr.cntl (user system programs)
- base/cntl/usr.cntl
  - delivered empty

MakeTPF Build Solution: Control Files - Records

- Data records
  - Contain 32 semicolon delimited fields
- Include records
  - Contain: include pathname/control_filename
  - Supported one level deep only
  - Allow programs to optionally be grouped into sub-control files
- Comment records
  - Begin with #
- All three types of records can exist in a top level control file
  - Only comment and data records can exist in an included control file

MakeTPF Build Solution: Control File Fields

1. Program name
   - For Standard Libraries, RT, CIMRs, KPTS, and Transfer Vectors, this is a 4 character program name, in upper case
   - For Offline z/OS programs, this is a max. of 6 characters and should be specified in upper case
   - For Offline Linux programs, there is no length limit and program names are case sensitive
   - In all cases, the name specified must match the value specified for the primary target in the program makefile.

2. Program type
   - APP - Real time program
   - ARCHIVE - Archive (not loaded)
   - CIMR - CIMR component
   - CTKX - Keypoint X
   - GFKPT - General File KEYPOINT
   - IPL - IPL component
   - KPT - KEYPOINT
   - OL - Offline Program (EXE)
   - STUB - STUB only entry
   - UTIL - Utility program (EXE)

MakeTPF Build Solution: Control File Fields

3. Makefile name
   - the path and filename of the makefile needed to build the program.
   - the pathname is specified relative to the TPF root directory used in the build. For example, base/rt/ctis.mak

4. Number of passes required for build
   - 0 Program is not built; used for stub-only entries
   - 1 Program is only built on the first pass of the build, with link resolution performed. Assumes libraries required for external link resolution appear before this program in the control file.
   - 2 Program is built on both passes. On the first pass the program is assembled/compiled, and linked without verification of external references. This is so the lib will be generated and dependent programs that come after it in the build will link. On the second pass, the program is re-linked with verification of external references.

MakeTPF Build Solution: Control File Fields

5. Program system allocation code
   - BSS - Program is only allocated in the BSS
   - ALL - Program is allocated in all subsystems
   - EXC - Program is allocated in all subsystems except the BSS
   - SHARED - Program is only allocated in the BSS
   - Linux - Program is a Linux offline/utility
   - OS/390 - Program is a z/OS offline
   - ssname - Named BSS or SS name program is to be allocated in

6. Object shippable indicator
   - NOOBJ - executable is system dependent, cannot be shipped
   - OBJ - executable is system independent, can be shipped
   - OCO - executable is object code only, must be shipped
   - LGPL - executable is derived from open source, cannot be shipped

MakeTPF Build Solution: Control File Fields

7. Function switch
   - Identifies whether or not the program is to be built, based on the system configuration.
   - Switch values must correspond to the name of an environment variable defined during the build (via the cfg file).
   - The variables must have a value of ON for the line to be processed in the cntl file.
   - If no switch applies, the value TPF_SBALL must be supplied.

8. LOAD indicator
   - LOADGF - The program is to be loaded in General File Only
   - LOAD - The program is to be loaded in both GF/Online
   - LOADONLINE - The program is to be loaded online only
   - IPAT - The program is to be input to salo/ipat only
   - NOLOAD - The program is not to be loaded

MakeTPF Build Solution: Control File Fields

9. STUB indicator
   - STUB - A stub is generated
   - NOSTUB -No stub is generated

10. FETCH indicator
   - PRELOAD - The program is brought in before cycle up
   - DEMAND - The program is not brought in until it is called
   - DEFAULT - The program is brought in during restart if the system is not in test mode, otherwise treated like DEMAND

11. DEBUG TRACE indicator
   - DEBUG - The program can be traced with the TPF debugger
   - NODEBUG - The program cannot be traced with the TPF debugger

MakeTPF Build Solution: Control File Fields

12. TIMEOUT value

- CALLER - The program inherits the timeout characteristics of the program that called it. If the calling program has its timeout value set to CALLER, the current program inherits the timeout characteristics of the first previous calling program that is not defined as CALLER. If a program has the CALLER option but is not called by any other program, the default (DEFAULT) timeout is used.
- DEFAULT - The program will use the default timeout specified in keypoint A.
- interval- A number (from 5 to 65535) of 10-millisecond intervals that a program can run without giving up control before system error 000010 is issued.

MakeTPF Build Solution: Control File Fields

13. AFFINITY indicator

- PROGRAM - an entry running in the program will only be processed by the I-stream engine that the entry running on when it first enters the program and will not be dynamically balanced by the scheduler
- NONE - an entry running in the program can be dynamically balanced by the scheduler

14. DUMP GROUP Name

- Identifies the dump group name
- A 1-8 character name is expected

15. TRACE GROUP Name

- Identifies the trace group name
- A 1-8 character name is expected

MakeTPF Build Solution: Control File Fields

16. BYPASS AUTHORIZATION indicator
    - BPAUTH - The program has bypass authorization
    - NOBPAUTH - The program does not have bypass authorization
17. RESTRICTED MACRO indicator
    - RESTRICT - The program can use restricted macros
    - NORESTRICT - The program cannot use restricted macros
18. MONTC MACRO indicator
    - MONTC - The program can use the MONTC macro
    - NOMONTC - The program cannot use MONTC macro

MakeTPF Build Solution: Control File Fields

19. KEY0 MACRO indicator

- KEY0 - The program can store into protected storage
- NOKEY0 - The program cannot store into protected storage

20. COMMON BLOCK indicator

- CMB - The program can issue the macro to get common blocks
- NOCMB - The program cannot issue the macro to get common blocks

21. - 28. User Authorization Fields

- Supported values must correspond with settings in c_idsalo.h
- Note: Value translated to uppercase before passing to salo

29. - 32. User Fields

- Field reserved for customer use

MakeTPF Build Solution: Control Files

	APP	CIMR	KPT	GFKPT	CTKX	IPL	ARCHIVE	STUB	UTIL	OL
NAME	X	X	X	X	X	X	X	X	X	X
TYPE	X	X	X	X	X	X	X	X	X	X
MAKEFILE	X	X	X	X	X	X	X	X	X	X
PASSES	X	X	X	X	X	X	X		X	X
SYSALLOC	X	X	X	X	X	X	X		X	X
OBJSHIP	X	X	X	X	X	X	X		X	X
FUNCSW	X	X	X	X	X	X	X		X	X
LOAD	X	X	X	X	X	X	X		X	X
STUB	X									
FETCH	X									
DEBUG	X									
TIMEOUT	X									
AFFINITY	X									
DUMPGRP	X									
TRACEGRP	X									
BPAUTH	X									
RESTRICT	X									
MONTC	X									
KEY0	X									
CMB	X									
USRAUTH	X									
USR										

System Generation: Setup and SIP

- Install source code
- Code the SIP deck
- Code the record ID attribute table
- Build Linux utilities
  - `bldtpf -util /ztpf/base/cntl/tpf.cntl`
- Perform the SIP
  - Assemble the face table object and generate the face table system configuration dependent macros and source files:
    - `bldtpf -fctb_src /ztpf/bss/src/sip.asm`
  - Assemble the SIP deck and generate the system configuration dependent macros and source files:
    - `bldtpf -sip /ztpf/bss/src/sip.asm`
  - Complete the assembly and link of the face table:
    - `bldtpf -fctb /ztpf/bss/src/sip.asm`

System Generation: Build

- Perform the Build
  - Build z/TPF online programs
    - `bldtpf -tpf /ztpf/base/cntl/tpf.cntl`
  - Build z/TPF offline Linux programs
    - `bldtpf -ol /ztpf/base/cntl/tpf.cntl`
  - Build z/TPF offline z/OS programs (run on z/OS)
    - `bldtpf -ol /ztpf/base/cntl/tpf.cntl`

Sample Customer Applications: Directory Structure

- billing/
  - include/
  - lib/
  - load/
  - lst/
  - macro/
  - obj/
  - src/

Sample Customer Applications: Environment File

- Filename:

  tpftools/include_ztpf_user/maketpf.env_billing

- Contents:

```
ROOTLIBDIRS := $(foreach d,$(APPL_ROOT),$d/billing/lib)
ROOTLOADDIRS := $(foreach d,$(APPL_ROOT),$d/billing/load)
ROOTEXPDIRS := $(foreach d,$(APPL_ROOT),$d/billing/exp)
ROOTOBJDIRS := $(foreach d,$(APPL_ROOT),$d/billing/obj)
ROOTLSTDIRS := $(foreach d,$(APPL_ROOT),$d/billing/lst)
ROOTINCDIRS := $(foreach d,$(APPL_ROOT),$d/billing/include)
ROOTMACDIRS := $(foreach d,$(APPL_ROOT),$d/billing/macro)
ROOTASMDIRS := $(foreach d,$(APPL_ROOT),$d/billing/src)
ROOTCPYDIRS := $(foreach d,$(APPL_ROOT),$d/billing/src)
ROOTCDIRS := $(foreach d,$(APPL_ROOT),$d/billing/src)
ROOTCPPDIRS := $(foreach d,$(APPL_ROOT),$d/billing/src)
```

Sample Customer Applications: Makefile

```
APP := B123

maketpf_env := billing
maketpf_env += base_rt

C_SRC := bill_amount.c
C_SRC += bill_date.c

include maketpf.rules
```

Sample Customer Applications: Configuration File

```
APPL_ROOT := /home/lafer/mywork
APPL_ROOT += /cust/apps

TPF_ROOT := /ztpf

TPF_BSS_NAME := BSS

CFLAGS_USER := -O0
```

Preparing for z/TPF: Define Your Directory Structure

- For those that do not yet have an hfs defined, this is your opportunity for a fresh start
- For those that already have an hfs structure defined for TPF 4.1 applications, you can still change the structure for z/TPF, using maketpf environment files to handle the different directory structures
- Considerations
  - Multiple application directories, divided by function
  - Sub-directories set up by logical groupings of programs rather than source file type
- One size does not fit all ...

Preparing for z/TPF: Set up MakeTPF 4.1

- Install MakeTPF for TPF 4.1
  - Same interfaces as MakeTPF for z/TPF
  - Rules are structured for DLM, DLL, LLM, and BAL programs
  - Version 2 to be available by year end (12/04) -- changes were made from MakeTPF Version 1 made available in 1Q04, to be more consistent with the z/TPF version
  - Available from the TPF Downloads website: http://www.ibm.com/tpf/downloads/tools.html
- Set up environment files
  - Map your TPF 4.1 hfs structure to the MakeTPF schema variables and setup up application environment files
  - Additional considerations are required with MakeTPF for TPF 4.1 to address datasets

Preparing for z/TPF: Create Makefiles

- Convert build scripts to makefiles
  - Tools are provided to assist with converting build scripts to MakeTPF format makefiles
  - These makefiles will work with both the TPF 4.1 and z/TPF versions of MakeTPF.
    - The interface is the same.
    - The rules that are run are different.
    - The environment files handle the differing directory structures for the TPF 4.1 and z/TPF 1.1

Preparing for z/TPF: Consider Using The TPF Toolkit

- MakeTPF for TPF 4.1 can be used with the TPF Toolkit for Websphere studios, as can MakeTPF for z/TPF
- May help ease the transition between TPF 4.1 and z/TPF by sharing the same interface
- Can help isolate developers from the implementation

Preparing for z/TPF: Education

- Learn the Unix/Linux basics
  - Directory/file management
  - Editors
  - Managing file permissions
  - Finding and searching
- Learn a shell language
  - Useful for automation

Legal

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, z/OS, Websphere, DB/2, z/VM, 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.

Linux is a trademark of Linus Torvald in the United States, other countries, or both.

Windows is the trademark of Microsoft Corporation in the United States, other countries,or both.

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