z/TPF EE V1.1
z/TPFDF V1.1
TPF Toolkit for WebSphere® Studio V3
TPF Operations Server V1.2

IBM Software Group

TPF Users Group Spring 2006

Best Practices for Migrating Your TPF4.1 Applications

C/C++ Migration to z/TPF

Name : Bob Kopac
Venue : Application Development Subcommittee

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Steps for C/C++ Migration to z/TPF

Use the TPF Toolkit or functional equivalent for ease of conversion of directories or of a program

1. Apply automated replacement rules on all code
   - This should handle the majority of changes
2. Apply remaining rules to header files first
3. Apply remaining rules to segments
4. Compile using z/OS compiler
5. Compile using GCC
6. Build, and test on TPF4.1
7. Build, and test on z/TPF

C/C++ Code Modification Rules

1. Rules for automated simple replacement
2. Changes identified by compiler messages
3. Rules that require investigation
4. Less common rules

No Source Changes Needed

- C programs calling assembler
  - No changes when all C/C++ user programs are below 2 GB

- Enumerated types
  - TPF4.1: 1, 2, or 4 bytes in size based on values
  - z/TPF: 4 bytes, unless a GCC option specified
    - Compile using the -fshort-enums option

TPF Toolkit Rules for Automated Simple Replacement

Scan, auto-correction

Note: For the TPF Toolkit, if 2 or more rules affect the same line (overlap), you may have to rescan and auto-correct.

Rule Name	TPF Toolkit Rule description
OTRTRIGa	Trigraph usage should be avoided
PJ29593a	Use tpf header files in ***tpf*** subdirectory
PJ29593b	Remove $ symbol from all header file names -- $ incompatible with Linux
PJ29593c	Update #include statements to use _ instead of $ in header file names

TPF Toolkit Rules for Automated Simple Replacement (continued)

Rule Name	TPF Toolkit Rule description
PJ29593d	Use angle brackets < > in #include statements for TPF headers
PJ29595e	cs() (compare-and-swap) function is no longer located in stdlib.h header. Now in tpf/cmpswp.h
PJ29595f	cds() (compare-and-double-swap) function is longer located in stdlib.h header. Now in tpf/cmpswp.h
PJ29937a	Some system headers moved to sys subdirectory
OTRTRWTa	Continued strings cannot have trailing whitespace

Example of TPF Toolkit Rules for Simple Replacement

Code Before

```
...
??= include <stdlib.h>
#include "c$eb0eb.h"
#include <sysgtime.h>

...
cs(&xx, &yy, i);
```

Code After

```
...
#include <stdlib.h>
#include <tpf/c_eb0eb.h>
#include <sys/time.h>
#include <tpf/cmpswp.h>
...
cs(&xx, &yy, i);
```

TPF Toolkit Rules for Automated Simple Replacement (continued)

OTRKYWDc z/OS specific keyword _Export is not needed for z/TPF

```
#ifdef __370__
    _Export
#endif
```

TPF Toolkit Rules for Automated Simple Replacement (continued)

OTRPACKa — *#pragma pack()* statements can have different meaning on TPF 4.1 and z/TPF

Before change

```
#pragma pack(packed)
#pragma pack(twobyte)
#pragma pack(full)
#pragma pack()
```

After change

```
#pragma pack(1)
#pragma pack(2)
#pragma pack(4)
#pragma pack(4)
```

TPF Toolkit Rules for Automated Simple Replacement (continued)

OTRPACKb *#pragma pack(reset)* is not supported on GCC

Before

```
#pragma pack(reset)
```

After

```
#ifdef __370__
  #pragma pack(reset)
#elseif
  #pragma pack()
#endif
```

GCC: *#pragma pack()* returns to natural alignment

z/OS: *#pragma pack()* is 4-byte alignment
#pragma pack(reset) returns alignment to previous rule

TPF Toolkit Rules for Automated Simple Replacement (continued)

OTRPACKc

_*Packed* option should be surrounded with
#ifdef __370__ _Packed #endif
For GCC, use *__attribute__((packed))*

```
typedef
#ifdef __370__
     _Packed
#endif
 struct {
   int mdata;
   char  mtext[1024];
    }
#ifndef __370__
     __attribute__((packed))
#endif
 MyType;
```

TPF Toolkit Rules for Automated Simple Replacement (continued)

OTRLONGa	Use *int* data type instead of *long* when declaring variables
OTRLONGb	Use *int* data type instead of *long* when casting variables

These rules should be performed

- Reason:
  - *long* data type size is different on TPF4.1 than on z/OS:
    - 4 bytes on TPF4.1 using z/OS compiler
    - **8** bytes on z/TPF using GCC
  - *int* data type size is 4 bytes using both z/OS compiler and GCC

Changes Identified by GCC Messages

See the "Common errors and warnings" section of the **z/TPF Migration Guide**

Examples:

- Use parentheses ( ) to explicitly group operations

```
if (a || (b && c))
```

  - OTRPRECa TPF Toolkit Rule can be used to find if desired
- Braces { } must be used when initializing arrays of structures
  - OTRARINa TPF Toolkit Rule can be used to find if desired
- Use braces { } around nested if statements

```
if(a){if(b) QZZ1();} else QZZ2();
```

- offsetof( ) must have TYPE data type as 1st parameter
  - GCC does not allow variable of type TYPE as z/OS does

Changes Identified by Compiler Messages

- Change *int* back to *long* when the variable is used to store a pointer
  - GCC warning message
  - z/OS compiler severe error occurs for argument passing by reference

TPF Toolkit Rules That Require Investigation

PJ29575a Flag all pointers. Pointer size changes from 32-bit to 64-bit on z/TPF

- Load all user programs and TPF libraries used by them below 2 GB
- All pointer values then will fit in the low-order 31 bits of the 64-bit pointer with the high-order bits being zeroes
- The values can be stored in a 32-bit storage area
  - Stack, Heap, ECB private area, static, literal data pointers

TPF Toolkit Rules That Require Investigation (continued)

PJ29575a (continued)

- Change all pointers in user programs to the equivalent 32-bit pointers

__ptr32_t	Use to declare 32-bit void pointers
__chptr_t	Use to declare 32-bit char pointers
__uiptr32_t	Use to declare 32-bit unsigned int pointers
PTR32ATT definition	Use for other pointer types

Example:

Code Before	Code After
void * vp;	__ptr32_t vp;
struct mystruct * msp;	struct mystruct PTR32ATT * msp;

TPF Toolkit Rules That Require Investigation (continued)

- Change *time_t*, *size_t*, and *ssize_t* to *time_t32*, *size_t32*, and *ssize_t32*

  - PJ29630a Types *time_t*, *size_t* and *ssize_t* changed from 32-bit to 64-bit

  - PJ29630b Function calls passing types *time_t*, *size_t* and *ssize_t* may require changes
    - Need to change only for calls-by-reference
    - PJ29630b is defaulted to be OFF
    - If needed, scan with only this rule turned on

TPF Toolkit Rules that require investigation (continued)

- for z/TPF, a 32-bit pointer is equivalent to a regular pointer only by value and not by reference
- If you pass the address of a 32-bit pointer where the address of a 64-bit pointer is expected, the called function may take a run-time error
  - Also applies for address of *size_t32*, *ssize_t32*, *time_t32*
- Assign the 32-bit variable to a local 64-bit variable before passing address of the local variable

```
size_t32  X = 8;
size_t  localbit64;
...
localbit64 = X;   /* to 64-bit field*/
myfunc(&localbit64);
```

TPF Toolkit Rules that require investigation (continued)

- Assembler calling a C function
  - When no arguments are passed, no changes are required
  - When arguments are passed and the function is called by name, z/TPF APAR PJ31214 will simplify changes
    - No change needed to the calling assembler programs
    - Create CPROCs for each of these C programs and put in a specified macro
    - ENTRC and ENTNC macros will use this macro to detect a call to a 'C program with parameters' and will pass the parameters appropriately
    - This assumes R1 points to a valid parameter list

TPF Toolkit Rules that require investigation (continued)

Assembler-written C functions

- Use TMSPC/TMSEC for initial migration
- Must use PRLGC/EPLGC instead of TMSPC/TMSEC when:
  - Passing more than 5 arguments, or
  - Passing floating point values
  - Note: No additional changes needed when all C/C++ user programs are below 2 GB
- No TPF Toolkit rules to find these
- Examine prototypes

TPF Toolkit Rules that require investigation (continued)

Assembler-written C functions

- TMSPC/TMSEC
  - The following TPF Toolkit Rules detect code that has been migrated to ISO-C from TARGET(TPF) code

PJ29640a	TMSPC parameter MIGRATION=YES is obsolete on z/TPF
PJ29640b	Copy of program parameters in R1
PJ29640c	Copy of stack pointer in R13
PJ29640d	MIGRATION= keyword not valid on TMSPC macros on z/TPF - Detects MIGRATION=NO

TPF Toolkit Rules that require investigation (continued)

- CE3SPTR is obsolete in z/TPF

PJ29640c Use R13 to reference the stack pointer
PJ29640e Use CSTKC to save the C stack frame pointer
PJ29640f Use CSTKC to restore the C stack frame pointer

PJ29640g CE3SPTR is obsolete on z/TPF (manual change)
  - Replace all remaining references to CE3SPTR not changed by PJ29640e and PJ29640f

TPF Toolkit Rules that require investigation (continued)

- CSTKLBAS is obsolete on z/TPF

PJ29640h, PJ29640i, PJ29640k, PJ29640l

- Use PBASC to save and restore the program base

PJ29640j CSTKLBAS obsolete on z/TPF (manual change)

- Replace all remaining references to CSTKLBAS not changed by PJ29640h and PJ29640i

TPF Toolkit Rules that require investigation (continued)

- Use the *sizeof* operator to determine the size of pointers

For example:

```
malloc(100 * 4);                    /* incorrect */
malloc (100 * sizeof(long *));/* correct */
```

OTRHRCDa Hardcoded calculations based on 4-byte pointers must account for 8-byte pointers

- This rule flags **all** occurrences of the character "4"
  - This rule is defaulted to be OFF
  - If needed, scan with only this one rule turned on
- Will not find other coding that may multiply by 4

TPF Toolkit Rules that require investigation (continued)

OTRKYWDb z/OS specific keywords must be removed for z/TPF
_cdecl __cdecl __callback

- For example: void __cdecl f1();

OTRPRAGa Replace or remove #pragma directives

- For example: #pragma page
  #pragma title

OTRSEQNb Sequence numbers must be removed from C/CPP files for z/TPF

Less Common TPF Toolkit Rules

PJ29436a	Change EBW012 to EBW024 in programs activated by activate-on-receipt (AOR) calls
PJ29957a	setlocale function changed for obsolete category LC_TOD
PJ29957b	Replace usage of locale category LC_TOD with TZ environment variable support
PJ29980a	Remove usage of *long double* type
OTRDFRVa	Change functions defined using *#define* returning a value to static inline functions
OTRPRGCa	Change *progc* function calls that use PAT_PBI and PAT_DBI parameters

Less Common TPF Toolkit Rules

OTRLCALa	LC_SYNTAX category is obsolete on z/TPF
OTRLCALb	Non-standard collation functions from TPF 4.1 are not supported on z/TPF
OTRLCALc	Header file collate.h not supported on z/TPF
PJ29974a	Error trapping for log() and log10() changed on z/TPF -- Use tpf_chk_log_dbl() and tpf_chk_log10_dbl() to check for errors
OTRDRDTa	*clock_t* data type format changes from *double* to *long*
OTRWDCTa	Use wide characters for in-memory processing only (*wchar_t* changed to 4-byte Unicode)
OTRWDCTb	Wide characters coded using hex values break single source (literals)

Additional Migration Issues

- Floating point representation
  - TPF 4.1 supports HFP, GCC for z/TPF supports BFP
    - There are no Toolkit rules
    - TPF conversion functions exist:
      - tpf__fp_htob(), tpf__fp_btoh(),
      - tpf__fp_ntob(), tpf__fp_bton(),
      - tpf__fp_ntoh(), tpf__fp_hton()
- Header file names are case-sensitive on z/TPF
  - This was not true on TPF4.1 for PDSes
- Write any new TPF4.1 code according to guidelines in **z/TPF Migration Guide**

To Simplify C/C++ Migration to z/TPF

- Use automation as much as possible
- Use tool to detect as many rules as possible
- Use z/OS compiler and GCC messages to help identify additional changes

- Automation should handle the majority of changes
  - Allows better quality and speed of changes
  - Helps keep migration costs under control

Trademarks

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

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

Notes

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.