Part 2. Tasks ................................................................. 37

Planning for your migration .............................................. 39
Preparation for your environment ........................................ 43
Maintenance upgrade requirements .................................... 43
Generating the z/TPF system ............................................. 45
Modify your SIP input deck ............................................. 45
Changes to SIP macros .................................................. 45
Changes for the RAMFIL statements ................................... 47
Migrating to the z/TPFDF product ..................................... 51
Customizing the code .................................................... 53
Migrating your C library functions ..................................... 55
Migrating your application programs ................................. 57
Coexistence, migration, and fallback ................................ 63
Running TPF 4.1 and z/TPF ............................................ 63
Migrating from a TPF 4.1 system to a z/TPF system .......... 65
Falling back to a TPF 4.1 system ..................................... 66

Part 3. Reference ............................................................... 67

Operating environment requirements and planning information  69
Hardware ................................................................. 69
Servers ................................................................. 69
Storage devices ......................................................... 69
Interconnection devices ............................................... 70
Communication controllers ........................................... 70
Consoles ............................................................. 71
Loosely coupled complex ............................................. 71
High Performance Option (HPO) feature ............................... 71
Software (programming requirements) ................................. 71
Operating system requirements ........................................ 71
Library system requirements .......................................... 73
z/Virtual Machine requirements ....................................... 73
TPF Operations Server requirements .................................. 73
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNA-based communication requirements</td>
<td>73</td>
</tr>
<tr>
<td>TCP/IP-based communication requirements</td>
<td>74</td>
</tr>
<tr>
<td>Product information requirements</td>
<td>75</td>
</tr>
<tr>
<td><strong>Single source and migration APARs</strong></td>
<td>77</td>
</tr>
<tr>
<td>Single source APARs</td>
<td>77</td>
</tr>
<tr>
<td>Migration APARs</td>
<td>78</td>
</tr>
<tr>
<td><strong>Mapping data types.</strong></td>
<td>81</td>
</tr>
<tr>
<td><strong>Common errors and warnings</strong></td>
<td>83</td>
</tr>
<tr>
<td>Compiler errors</td>
<td>83</td>
</tr>
<tr>
<td>Header files not found</td>
<td>83</td>
</tr>
<tr>
<td>Parse error</td>
<td>83</td>
</tr>
<tr>
<td>Function coded as a #define</td>
<td>83</td>
</tr>
<tr>
<td>Initializer error</td>
<td>84</td>
</tr>
<tr>
<td>Ambiguous coding practices</td>
<td>84</td>
</tr>
<tr>
<td>Undefined data type</td>
<td>85</td>
</tr>
<tr>
<td>long to int conversion problems</td>
<td>85</td>
</tr>
<tr>
<td>Relocation error</td>
<td>86</td>
</tr>
<tr>
<td>stdio.h error</td>
<td>86</td>
</tr>
<tr>
<td><strong>Compiler warnings</strong></td>
<td>87</td>
</tr>
<tr>
<td>Casting a pointer to an integer</td>
<td>87</td>
</tr>
<tr>
<td>Class definitions and constructor variable parameters</td>
<td>87</td>
</tr>
<tr>
<td>Defining counters in a loop</td>
<td>88</td>
</tr>
<tr>
<td>Assigning pointer to integer without cast</td>
<td>88</td>
</tr>
<tr>
<td>Unused variables or functions</td>
<td>89</td>
</tr>
<tr>
<td>Return with no value</td>
<td>89</td>
</tr>
<tr>
<td>Control reaches the end of a non-void function</td>
<td>89</td>
</tr>
<tr>
<td>Format code does not match variable type</td>
<td>89</td>
</tr>
<tr>
<td>No function prototype</td>
<td>89</td>
</tr>
<tr>
<td>Backslash and newline</td>
<td>90</td>
</tr>
<tr>
<td>Enumeration value not handled</td>
<td>90</td>
</tr>
<tr>
<td>Problem with 2-digit year</td>
<td>90</td>
</tr>
<tr>
<td>Use of cast expressions</td>
<td>91</td>
</tr>
<tr>
<td>Type-pun problems</td>
<td>91</td>
</tr>
<tr>
<td>Attribute packed statement ignored</td>
<td>92</td>
</tr>
<tr>
<td>Problem generating the correct code</td>
<td>92</td>
</tr>
<tr>
<td><strong>Run-time errors</strong></td>
<td>93</td>
</tr>
<tr>
<td>DLM-style entry point</td>
<td>93</td>
</tr>
<tr>
<td>Offsets do not match</td>
<td>93</td>
</tr>
<tr>
<td>CTL-4 system error</td>
<td>94</td>
</tr>
</tbody>
</table>
Figures

1. Virtual storage layout: part 1
2. Virtual storage layout: part 2
## Tables

1. z/TPFDF C functions and parameters that can affect performance. ................. 19  
2. Code migration tools ................................................................................. 34  
3. New records ............................................................................................... 47  
4. Records that are no longer supported. .......................................................... 48  
5. Increased records ....................................................................................... 50  
6. Decreased records ..................................................................................... 50  
7. TPFDF 1.1.3 APARs beyond PUT 16 with migration considerations ............. 52  
8. Old SPM names and replacements ............................................................... 61
Part 1. Concepts
Guidelines for using this information

This information is for any customer who is migrating, or planning to migrate, from:

- Transaction Processing Facility Version 4 Release 1 (TPF 4.1) to z/Transaction Processing Facility Enterprise Edition Version 1 Release 1 (z/TPF)
- TPF Database Facility Version 1 Release 1 Modification Level 3 (TPFDF 1.1.3) to z/TPF Database Facility Enterprise Edition Version 1 Release 1 (z/TPFDF).

This information describes the benefits of the z/TPF system and the z/TPFDF product, and provides general information to help you plan for your migration. See z/TPF Concepts and Structures for a comprehensive overview of the z/TPF system and the z/TPFDF product. For specific information about installation and system generation, see:

- z/TPF Program Management
- z/TPF System Installation Support Reference
- z/TPF and z/TPFDF System Generation

Getting Started

- For information about the benefits of the z/TPF system and z/TPFDF product, see “z/TPF system and z/TPFDF product benefits” on page 5.
- For information about the changes made to the z/TPF system and z/TPFDF product, see “z/TPF system and z/TPFDF product changes” on page 9.
- To begin your migration, see the planning checklist in “Planning for your migration” on page 39.
**z/TPF system and z/TPFDF product benefits**

The z/TPF operating system works with application programs to process transactions in a real-time environment. For example, you can use the z/TPF system for computing needs such as transaction processing, network switching, and front-end processing. The z/TPF system replaces earlier versions of the TPF system. If you require the z/TPF functions, you must migrate to the z/TPF system.

The z/TPF system is for businesses and organizations that have high online transaction volumes and large networks. Because of its high-capacity and high-availability characteristics, the z/TPF system is well-suited for environments where growth is expected to be very fast or unpredictable, or where there are high peak periods of transaction activity. It is especially useful for application programs that need high capacity and extremely low cost per transaction.

The z/TPFDF product is a database manager for application programs that run in a z/TPF operating environment. To increase the productivity of application programmers, the z/TPFDF product provides:

- A logical method to organize the database
- A set of standardized assembler macros or C functions that form the application programming interface (API)
- Central routines to access and manage the database
- Utilities for database maintenance and testing.

After you install the z/TPFDF product, z/TPFDF application programs are no longer sensitive to the physical implementation of the database.

The z/TPF system and the z/TPFDF product provide benefits in the following areas:

- "Common environment"
- "Application productivity" on page 6
- "Availability" on page 7
- "Lifting constraints" on page 7
- "Coexistence" on page 7

See "z/TPF system and z/TPFDF product changes" on page 9 for more detailed information.

**Common environment**

The majority of the development and build environment for the z/TPF system and z/TPFDF product has been migrated to the model that is used by the open source community; that is, the z/TPF system uses a Linux-based development and build environment, which is a well-documented, well-defined environment and has many tools available to help with development and building.

By using the GNU compiler collection (GCC) and support for common (open) programming protocols, the z/TPF system can share applications, tooling, and development infrastructure with Linux, which is considered one of the most common open systems available. This allows you to obtain and use large quantities of easily ported open software. Using these common programming standards can help you to lower the overall cost of application programming by allowing you to use common programming skills and providing the ability to port existing code.
The move to a Linux-based build environment and the GCC means that you are no longer required to submit your programs to z/OS for assembly or compilation. The GCC is a true cross compiler, and as such, can be run on Linux anywhere from the desktop to the mainframe (under z/VM, for example).

The z/TPF system provides a new, makefile-based build process called the MakeTPF build solution that makes it much easier to build programs and prevents common mistakes. This new build process also makes program update tape (PUT) upgrades easier.

A development workbench helps programmers to pull all the pieces of this development environment together. The IBM TPF Toolkit for WebSphere Studio, which is the follow-on to VisualAge TPF for Microsoft Windows NT, supports both the TPF 4.1 system and the z/TPF system simultaneously and represents the next generation of application development technology to help you build applications faster and easier.

Additional information:
- See “Build environment” on page 28 and [z/TPF Program Management](http://www.ibm.com/software/awdtools/tpftoolkit/index.html) for more information about the MakeTPF build solution.

**Application productivity**

The enhancements that are included with the z/TPF system and z/TPFDF product provide application programmers with the ability to incorporate changes that can help improve the functionality, portability, usability, and performance of the applications suite.

The following examples describe some of the new features that can help improve the productivity of your application programming staff and the reliability of your system:

- A common challenge for application programmers is how to manage the use of program registers by the program. With the z/TPF system, there are additional registers that are available for the application program to use.
- Assembler programs now can use the program stack (called the application stack) that was previously only available for C programs.
- Assembler programs now can call a C function directly by using the CALLC macro.
- The amount of main storage that is available for the application program to use is greatly increased. You also can assign a unique token to an area of system heap storage, and you can use this token later to find the area of storage by name instead of by address.
- The z/TPF debugger runs with code compiled with optimization. This lets you use the debugger against the same object that is used in the run-time production environment. The z/TPF debugger also can be used with the z/TPFDF product.
- Both ASCII and EBCDIC translations are displayed in data dumps, which simplify debugging efforts.

Additional information:
- See “Application programming” on page 24 for more information about application programming changes.
Availability

As in previous versions, the z/TPF system was designed from the start to provide high availability. The following list describes some of the enhancements that help to improve the availability of the system for this latest version:

- The dump buffer area (DBA) helps to reduce the time that the system stops while taking a dump; that is, the system stops only long enough to store the dump in memory.
- With the z/TPF system, enhancements such as physical block owner support and the ECB resource monitor help you to monitor the use of system resources so that you can make corrections before any problems occur.
- Support for prime/duplicate module pairing (also referred to as even/odd module pairing), along with new user exits for online reorganization, can make the database reorganization effort easier.

Note: Migration or coexistence between the standard IBM duplication module configuration and prime/duplicate module pairing is not supported.

- Enhancements were made in TCP/IP support to help with availability as well. For example, you now can define a TCP/IP application, socket, or both, as high priority so that its input messages go on the ready list instead of the input list. You also can define the discard priority for a TCP/IP application, socket, or both, which defines the order in which TCP/IP input messages are discarded when the system is running low on resources.
- In the TPF 4.1 system, get file storage (GFS) was not active in 1052 state. In the z/TPF system, you can use the ZPOOL 1052 command to enable GFS in 1052 state. This lets you run any applications and utilities that use GFS-type APIs in 1052 state.
- In the TPF 4.1 system, sockets cannot be started below CRAS state. However, in the z/TPF system, you can define a socket to be 1052-state capable so that the socket can be created in 1052 state. The SetTCP1052 function lets an application create a socket that is 1052-state capable. You also can define applications to the Internet daemon as 1052-state capable and you can define OSA-Express connections as 1052-state capable.

Lifting constraints

The z/TPF system provides the following constraint relief:

- The ability to use more than 2 GB of real storage
- Support for as many as 255 subsystem users (SSUs)
- Support for as many as 40 000 DASDs
- Support for as many as X'FFFF' symbolic device addresses (SDAs), including tape addresses.

Coexistence

With the High Performance Option (HPO) feature, you can have a TPF 4.1 system and z/TPF system running in the same complex. See “Coexistence, migration, and fallback” on page 63 for more information about coexistence considerations.
You also can maintain a single set of source for applications that need to run on both the TPF 4.1 and z/TPF systems.

**Note:** You must apply specific APARs, referred to as single source APARs, to your TPF 4.1 system to do this; and you cannot use the same source for applications on both levels of the system if you want to use any of the new z/TPF functionality. See "Single source and migration APARs" on page 77 for more information.
z/TPF system and z/TPFDF product changes

The following information provides an overview of the areas that are new, changed, and no longer supported in the z/TPF system and z/TPFDF product. Use this information with the procedures provided in Part 2, “Tasks,” on page 37 to migrate to these new releases.

- “z/TPF system structural characteristics”
- “z/TPF processing” on page 16
- “Data organization” on page 17
- “Database support” on page 18
- “Data communication” on page 24
- “Application programming” on page 24
- “User Exits” on page 28
- “Build environment” on page 28
- “Middleware and Web services” on page 29
- “Features” on page 29
- “Problem determination” on page 30
- “Performance” on page 32
- “Operations” on page 33
- “Migration tools” on page 34
- “No longer supported” on page 35.

z/TPF system structural characteristics

The following information provides an overview of the changes in z/TPF structural characteristics.

Loaders and images

A number of enhancements were made to the loaders for the z/TPF system.

- Programs are loaded only from a hierarchical file system (HFS) on Linux or z/OS. Therefore, the names of the executable are case sensitive and the file extension, if one exists, must match. The paths that the loader will look for the programs can still be specified in a particular search order, but you can specify the path in new ways that provide more flexibility.

- The format of the loader input file, referred to as the load deck in the TPF 4.1 system, has changed.

- The offline loader can be run on z/OS or on Linux.

- The program allocator table (PAT) is now referred to as the program attribute table (PAT). In addition, IBMPAL has been replaced by the tpf.cntl file and SPPGML has been replaced by the tpf.cntl file and the program makefiles.

- All programs, including basic assembler language (BAL), are stored on file using both #PROG and #XPRG records. In the TPF 4.1 system, only C and C++ programs used #XPRG records. Because the layout of programs on file has changed, the FILE parameter of the GETPC general macro is no longer supported.

- In the TPF 4.1 system, program version records (PVRs) let you store information about the version history for a program. PVRs are no longer supported in the z/TPF system. Instead, you now can put version information in the #PROG record.
Changes have been made to the way that you allocate programs dynamically. In the TPF 4.1 system, you had to create a new program allocation table offline and allocate the new programs online by entering the ZOLDR LOAD command with the PATU parameter specified. In the z/TPF system, you can allocate new programs dynamically by entering the ZAPAT ADD command.

You can display program listings online by using the ZDPGM command with the LISTING parameter specified.

Additional information:
- See z/TPF Program Management for more information about loaders.
- See “Coexistence, migration, and fallback” on page 63 for coexistence considerations.

External storage

The following information describes changes in external storage for the z/TPF system.

Tape blocking

The z/TPF system supports block sizes from 32 760 bytes to 128 KB. Use the ZTLBL command to set the block size for a tape. You also can use the ZTMNT command to override the block size. With these changes come the following considerations:

- RTA and RTL tapes must be blocked as follows:
  - The block size for the RTA tape is 32 760 bytes
  - The block size for the RTL tape is 128 KB.
- You can no longer force a dump from the RTL tape to the RTA tape if the RTL tape becomes unavailable. Therefore, you must have an RTL standby or ALT tape mounted if the RTL tape becomes unavailable.

Additional information: See z/TPF Operations for more information about the ZTLBL, ZTMNT, and other tape support commands.

Main storage

The z/TPF main storage layout supports the 64-bit architecture and larger available memory. Figure 1 on page 11 and Figure 2 on page 12 show the revised virtual storage layout.
<table>
<thead>
<tr>
<th>Protection key</th>
<th>Storage area</th>
<th>Storage area</th>
<th>Protection key</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>SVM DAT tables for application I-streams</td>
<td>SVM DAT tables for application I-streams</td>
<td>F</td>
</tr>
<tr>
<td>9</td>
<td>64-bit system heap</td>
<td>64-bit system heap</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>Preallocated 64-bit system heap</td>
<td>Preallocated 64-bit system heap</td>
<td>9</td>
</tr>
<tr>
<td>F</td>
<td>System heap control table</td>
<td>System heap control table</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>Dump buffer area</td>
<td>Dump buffer area</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>Physical block control tables (CCT, FCT, MFCT, and SCT)</td>
<td>Physical block control tables (CCT, FCT, MFCT, and SCT)</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>EVM DAT</td>
<td>EVM DAT</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>1-MB frames</td>
<td>64-bit ECB heap</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ECB trace tables</td>
<td>ECB trace tables</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Preallocated storage for 31-bit ECB heap, application stack, and ECB private area</td>
<td>Preallocated storage for 31-bit ECB heap, application stack, and ECB private area</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>SVAT storage</td>
<td>SVAT storage</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>64-bit CRPA</td>
<td>64-bit CRPA</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>IP message table</td>
<td>IP message table</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>Socket table</td>
<td>Socket table</td>
<td>F</td>
</tr>
<tr>
<td>64-bit VAAT user areas</td>
<td>64-bit VAAT user areas</td>
<td>user defined</td>
<td>64-bit VAAT user areas</td>
</tr>
<tr>
<td>4GB - 2GB</td>
<td>LDEV trace blocks</td>
<td>LDEV trace blocks</td>
<td>4GB - 2GB</td>
</tr>
<tr>
<td>4GB - 2GB</td>
<td>Main I-stream SVM segment and page tables</td>
<td>Main I-stream SVM segment and page tables</td>
<td>4GB - 2GB</td>
</tr>
<tr>
<td>4GB - 2GB</td>
<td>Main I-stream SVM region tables</td>
<td>Main I-stream SVM region tables</td>
<td>4GB - 2GB</td>
</tr>
</tbody>
</table>

**System virtual memory**

**ECB virtual memory**

*Figure 1. Virtual storage layout: part 1*
Key changes to main storage include the following:

<table>
<thead>
<tr>
<th>Protection key</th>
<th>Storage area</th>
<th>Protection key</th>
<th>Storage area</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-4 GB not mapped in SVM</td>
<td>31-bit system heap</td>
<td>2-4 GB not mapped in EVM</td>
<td>31-bit system heap</td>
</tr>
<tr>
<td>4GB</td>
<td></td>
<td>4GB</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4-KB frames</td>
<td>1</td>
<td>Preallocated ECB heap</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C environment (process scoped)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Preallocated application stack</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ECB stack area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 KB mapped not valid as stack guard</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ECB thread stack area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 KB mapped not valid as stack guard</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ECB private area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Preallocated ECB private area</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C environment (thread scoped)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>EVM ECB pages 1-3</td>
</tr>
<tr>
<td>1,2,1</td>
<td>ECBs</td>
<td>1,2,1</td>
<td>EVM ECB pages 1-3</td>
</tr>
<tr>
<td>F</td>
<td>SWBs</td>
<td>F</td>
<td>SWBs</td>
</tr>
<tr>
<td>F</td>
<td>IOBs</td>
<td>F</td>
<td>IOBs</td>
</tr>
<tr>
<td>F</td>
<td>4-KB common blocks</td>
<td>F</td>
<td>4-KB common blocks</td>
</tr>
<tr>
<td>F</td>
<td>PAT/EPLT</td>
<td>F</td>
<td>PAT/EPLT</td>
</tr>
<tr>
<td>1</td>
<td>31-bit CRPA</td>
<td>1</td>
<td>31-bit CRPA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>31-bit VAAT user areas</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>user defined</td>
</tr>
<tr>
<td>9,1,9</td>
<td>31-bit I-stream unique globals GL1, GL2, and GL3</td>
<td>9,1,9</td>
<td>31-bit I-stream unique globals GL1, GL2, and GL3</td>
</tr>
<tr>
<td></td>
<td>GL1, GL2, and GL3</td>
<td></td>
<td>GL1, GL2, and GL3</td>
</tr>
<tr>
<td>9,1,9</td>
<td>31-bit shared globals GL1, GL2, and GL3</td>
<td>9,1,9</td>
<td>31-bit shared globals GL1, GL2, and GL3</td>
</tr>
<tr>
<td></td>
<td>GL1, GL2, and GL3</td>
<td></td>
<td>GL1, GL2, and GL3</td>
</tr>
<tr>
<td>F</td>
<td>FACE, RIAT, SSUT, I-stream status table, and system keypoints allocated by IPLB</td>
<td>F</td>
<td>FACE, RIAT, SSUT, I-stream status table, and system keypoints allocated by IPLB</td>
</tr>
<tr>
<td>F</td>
<td>CIMR components</td>
<td>F</td>
<td>CIMR components</td>
</tr>
<tr>
<td>F</td>
<td>CIO control tables</td>
<td>F</td>
<td>CIO control tables</td>
</tr>
<tr>
<td>F</td>
<td>LDEVs</td>
<td>F</td>
<td>LDEVs</td>
</tr>
<tr>
<td>F</td>
<td>Branch trace tables for all I-streams</td>
<td>F</td>
<td>Branch trace tables for all I-streams</td>
</tr>
<tr>
<td>F</td>
<td>Prefix areas for all I-streams</td>
<td>F</td>
<td>Prefix areas for all I-streams</td>
</tr>
<tr>
<td>F</td>
<td>IPLB-&gt;CCCTIN parameter area</td>
<td>F</td>
<td>IPLB-&gt;CCCTIN parameter area</td>
</tr>
<tr>
<td>F</td>
<td>IPLB/CCIO</td>
<td>F</td>
<td>IPLB/CCIO</td>
</tr>
<tr>
<td>F</td>
<td>Control Program</td>
<td>F</td>
<td>Control Program</td>
</tr>
<tr>
<td>F</td>
<td>Prefix area (0–8 KB)</td>
<td>F</td>
<td>Prefix area (0–8 KB)</td>
</tr>
</tbody>
</table>

Figure 2. Virtual storage layout: part 2
The format of the entry control block (ECB) has changed with the use of 64-bit addresses. Some ECB areas require an increase in size and have been moved to new tables.

ECB heap
The ECB heap area now consists of two areas:
- The existing ECB heap area is now referred to as 31-bit ECB heap and resides below the 2-GB bar.
- A new ECB heap area, referred to as 64-bit ECB heap, resides above the 2-GB bar. The 64-bit ECB heap is available to applications; however, it is intended to be used only for very large heap requests.

Preallocated storage
The ECB private area, 31-bit ECB heap, and application stack each have a preallocated storage area that is permanently assigned to each ECB. The initial value is defined with the CORREQ macro during the system initialization process (SIP). You can change the size of each of these areas online by entering the ZCTKA ALTER command.

Application stack
The application stack, previously referred to as the ISO-C stack, is used by assembler programs and C language programs.

System heap
Similar to the ECB heap, the system heap area now consists of two areas:
- The existing system heap is now referred to as 31-bit system heap and resides below the 2-GB bar.
- A new system heap area, referred to as 64-bit system heap, resides above the 2-GB bar.

Core resident program area (CRPA)
The CRPA now consists of two areas:
- The 31-bit CRPA resides below the 2-GB bar and is for programs that run in 31-bit addressing mode. Use the APSIZE31 parameter on the RAM macro to specify the size of the 31-bit CRPA.
- The 64-bit CRPA is above the 2-GB bar, is fixed, and always equals the size of the 1-MB frame area.

1-MB frame physical block type
The 1-MB frame physical block type provides backing storage for ECB virtual memory (EVM) addresses in the ECB heap areas and the application stack. This block type also provides backing storage for virtual address areas such as the 31-bit and 64-bit CRPAs and the 31-bit and 64-bit system heap areas.

Additional information:
- See z/TPF Concepts and Structures and z/TPF Main Supervisor Reference for more information about main storage.
- See z/TPF Application Programming for more information about the ECB and application stack.

Memory configurations
The z/TPF system lets you define a specific memory configuration for a physical processor. You can define as many as eight memory configurations for a processor. The z/TPF system matches a memory configuration to a physical processor by using the best fit principle; that is, it uses the memory configuration that requires the
most amount of storage after meeting the requirements of a minimum amount of both virtual file access (VFA) and 31-bit system heap. You can use the ZCTKA PREFER command to specify a specific memory configuration as preferred so that CTIN will try to use this configuration first. This way, a physical processor can run different logical processors at different times by using the same memory configuration.

**Additional information:**
- See [z/TPF Concepts and Structures](http://www.ibm.com), [z/TPF Main Supervisor Reference](http://www.ibm.com) and [z/TPF and z/TPFDF System Generation](http://www.ibm.com) for more information about memory configurations.
- See [z/TPF Operations](http://www.ibm.com) for more information about the ZCTKA PREFER command, as well as the other ZCTKA commands that you can use to manage memory configurations.

### Task dispatching

The tightly coupled scheduler has been updated to more efficiently calculate the resource use of a processor and minimize response time. In the TPF 4.1 system, ECBs were assigned to an I-stream by the tightly coupled scheduler and remained on that I-stream unless the application program specifically switched I-streams by using the SWISC system macro. In the z/TPF system, you can set the I-stream affinity of a program to indicate whether the scheduler can balance entries processed by the program dynamically. If you define AFFINITY=PROGRAM, the scheduler will not attempt to balance the ECBs running in this program dynamically and will not move them to other I-streams. If you define AFFINITY=NONE, entries processed by the program will be balanced dynamically by the scheduler.

Use the ZAPAT command to set the I-stream affinity of a program. An application program can override what is defined in the program attributes with the EISAC general macro or the `tpf_eisac` function.

For the initial migration to the z/TPF system, leave your application programs defined as AFFINITY=PROGRAM. After you have migrated, examine the most frequently run programs and determine which ones can be changed to AFFINITY=NONE.

**Additional information:** See [z/TPF Concepts and Structures](http://www.ibm.com) and [z/TPF Main Supervisor Reference](http://www.ibm.com) for more information about the scheduler and dispatching work.

### System error processing

System error processing has changed to make it easier to customize what gets dumped and to accommodate z/Architecture. Enhancements were made to make solving dumps easier, reduce the amount of storage needed in the dumps, and to display more information about dumps on the console.

The format of the dumps now supports the new architecture. Dumps now provide detailed program information at the time of the dump; such as, the executable name and the object file name. The format of the dump has changed to have hexadecimal information on the left side and EBCDIC or ASCII translation on the right side for most items.

The entire program is no longer included in the dump. Instead, the link map for the failing program and the 4 KB area surrounding the error or SERRC general macro.
will be included in the dump. Additionally, only heap storage that was in use at the
time of the error will be dumped unless you specify otherwise on the ZIDOT
command or SERRC macro. The heap information will indicate who received the
storage and what size was requested.

A number of changes help to make solving dumps easier; for example:

- Block owner information is displayed on the console for the out of storage
dumps.
- New parameters on the ZASER command allow you to customize the percentage
  of frames and system work blocks (SWBs) when they are included in the dump.
- You can customize the dump by using the OWNER parameter on the IDOTB,
  IDOTD, and IDOTM system macros or the ZIDOT command. This lets you dump
  storage based on the owner of the block.
- You can use the following keywords on the IDATG system macro to dump only
  the identified block types that are in use:
  - ISWBUSE
  - IFRMUSE
  - IECBUSE
  - ICMNUSE.

See the IDATB system macro for a list of all of the predefined keywords.

- Enhancements to the ZIDOT and ZASER commands also make dumps easier to
  solve.
- You can include the system heap in a dump by using a unique token name rather
  than an address on the LISTC general macro.
- A formatted version of the current z/TPFDF SW00SR block is now included in
dumps.

The following enhancements allow you to further customize dumps and make them
more efficient and easier to solve:

- You can create different named manual dump definitions that you can use with
  the ZDUMP command to dump predefined areas of storage. Use the ZIDOT
  command or the IDOTM macro to define these named manual dumps.
- Dump formatting extensions allow a BAL shared object (BSO) or C shared object
  (CSO) to be called during dump processing to determine additional areas to be
  dumped and formatted.
- You can group programs in dump groups to make it easier to define overrides for
  programs that require the common areas to be dumped. This lets you associate
  dump overrides on an application-wide basis instead of having to define an
  override for each program.
- The dump buffer area used to buffer system error dumps until they can be
  written to the system dump tape. This reduces the time that the system is
  paused while taking a dump. Use the CORREQ macro or ZCTKA ALTER
  command to set the size of the dump buffer area, and use the ZDBAI DISPLAY
  and ZDBAI DELETE commands to manage the dump buffer area.

Additional information:

- See z/TPF Main Supervisor Reference for more information about system error
  processing and error recovery.
- See z/TPF Program Development Support Reference for more information about
dump format and the dump buffer area.
Resource control

The ECB resource monitor provides a centralized facility that monitors how selected system resources for each ECB are used and detects and, optionally, stops an ECB that requests excessive amounts of monitored resources. Use the ZECBM command to turn on or turn off the ECB resource monitor. You also can use the ZECBM command to display or set resource limits for an ECB online, or you can use the ECBMC general macro or tpf_ecbmc function to query or set resource limits from your application program.

Additional information:
- See z/TPF Application Programming for more information about the ECB resource monitor.
- See the z/TPF C/C++ Language Support User's Guide for more information about the tpf_ecbmc function.
- See z/TPF General Services for more information about the ECBMC macro.
- See z/TPF Operations for more information about the ZECBM command.

z/TPF processing

The following information describes changes in z/TPF processing for the z/TPF system.

Copy-on-write facility

As programs are fetched by the z/TPF system, the program data (also referred to as static data) is page-protected to avoid accidental corruption. In the z/TPF system, when an application program needs to change this data, a facility called copy-on-write creates an ECB-unique copy of the changed data using 4-KB frames. This allows the program to update the data without affecting other programs.

Additional information: See z/TPF Main Supervisor Reference for more information about the copy-on-write facility.

Tightly coupled

The z/TPF system is designed to support as many as 86 I-streams.

Note: The number of I-streams that you can use is limited to the number of active I-streams installed in the configuration.

Architecture

One of the primary changes for the z/TPF system is the support for z/Architecture, which is a 64-bit architecture that is provided on the IBM Enterprise processors. Most of the architecture changes consist of expanding the size of certain architected structures, such as:
- Using new instructions that provide 64-bit operations
- Expanding the size of the program status word (PSW) from 64 bits to 128 bits
- Expanding the size of the general registers from 32 bits to 64 bits
- Expanding the size of the control registers from 32 bits to 64 bits
- Expanding the size of page-table and segment-table entries from 32 bits to 64 bits
Defining three new levels of virtual address translation tables, called region first, region second, and region third

Expanding the size of various address fields from 32 bits to 64 bits

Supporting the extended time-of-day (TOD) clock format with a size of 128 bits

Expanding the prefix area, previously referred to as the prefix page, from 4 KB to 8 KB and changing the format of the prefix area

Using format-2 indirect data address words (IDAWs) to access storage areas about the 2-GB bar.

Additional information:

- See [z/Architecture Principles of Operation](#) for instruction descriptions and more information about z/Architecture.
- See [High Level Assembler for MVS & VM & VSE Language Reference](#) for more information about the High Level Assembler Release 5.
- See [z/Architecture Reference Summary](#) for a summary of the information provided in the [z/Architecture Principles of Operation](#).

Data organization

The following information describes data organization changes for the z/TPF system.

Keypoints

With the z/TPF system, keypoint support allows keypoints to exceed 4 KB. New keypoints consist of a full track of 4-KB records; that is, a total size of 48 KB. (The 4-KB records are logically concatenated into a single keypoint record.) The following keypoints were expanded from 4 KB to 48 KB:

- Keypoint A (CTKA)
- Keypoint D (CTKD)
- Keypoint 0 (CTK0)
- Keypoint 2 (CTK2)
- Keypoint 3 (CTK3).

The actual keypoint size is in the logical record header.

You also can use keypoint 6 (CTK6) as a large keypoint (greater than 4 KB) to support greater than 3999 DASD. If you want to use CTK6 as a large keypoint, specify CTK6EXP=Y on the ONLFIL system initialization process (SIP) macro.

The following general macros work with both 4 KB and greater than 4 KB keypoints:

**GETKC** Gets a keypoint record.

**UPDKC** Updates a keypoint record.

**RELKC** Releases a keypoint record.

**Note:** If you have code that accesses CTK3, you must change that code to use these macros.

CTKD is now a user keypoint. In the TPF 4.1 system, this keypoint contained communications protocol definitions; however, because these older protocols are not supported in the z/TPF system, CTKD has been changed to be a user keypoint.
Notes:
1. Many keypoint values are not compatible between the TPF 4.1 system and the z/TPF system.
2. CTKA, CTKD, CTKE, CTK0, CTK2, and CTK3 will not be shared between the TPF 4.1 system and the z/TPF system on a processor.

Additional information:
• See z/TPF Main Supervisor Reference for more information about keypoints.
• See z/TPF General Services for more information about the GETKC, UPDKC, and RELKC macros.
• See z/TPF and z/TPFDF System Generation for more information about the ONLFIL macro.
• See "Migrating your application programs" on page 57 for more information about specific steps that are required for migrating your application programs.
• See “Coexistence, migration, and fallback” on page 63 for more information about keypoints when running both the TPF 4.1 system and the z/TPF system.

VFA

Virtual file access (VFA) is above the 2-GB bar. All VFA blocks are accessed with 64-bit addressing. To prevent increased processing time, changes were made to VFA delay filing, storage allocation in CTIN, and emergency unlock.

You also must define your program records (record ID X'00FF') as non-VFA candidates because all programs are now core resident. This prevents VFA resources from being used when the programs are loaded into core.

Additional information: See z/TPF Database User’s Guide for more information about VFA.

Database support

The following information describes changes to database support for the z/TPF system.

z/TPF Database Facility Enterprise Edition

z/TPF Database Facility Enterprise Edition (referred to as z/TPFDF) is now a corequisite for the z/TPF system. The following changes have been made to the z/TPFDF product:

• The offline process of the macro label set (MLS) procedures was removed and a user exit in the DFUEX macro was added so you can list the DSECTs with label information that must be loaded to the online system. See z/TPF and z/TPFDF System Generation for more information about creating an MLS database.
• You can run z/TPFDF utilities in 1052 state when get file storage (GFS) is active. See z/TPFDF Utilities for more information.
• Changes were made to the following ZUDFM commands:
  – ZUDFM DISPLAY command processing no longer displays the entire subfile.
  – The ZUDFM DEF command now provides an option bit summary display for the database definition (DBDEF). You can display the bit settings for a specific file, or option bit settings for all DBDEFs, or for DBDEFs that meet specific criteria.
The ZUDFM RESTRICT command now displays information only for the command whose restrictions were modified. You also can display the entire restriction table or just the restrictions for a single command. See [z/TPFDF Commands](#) for more information about the ZUDFM commands.

- With the ZUDFC STATUS command, you can display z/TPFDF data collection status.
- z/TPFDF recoup now displays a specific message when a DBDEF begins or ends processing.
- The maximum number of files that you can open simultaneously with one ECB now includes W-type files, so you might need to raise the limit.
- A user exit was added to allow user-defined equates for use by the z/TPFDF product; for example, for user-defined algorithms. See [z/TPF System Installation Support Reference](#) for more information about z/TPFDF user exits.
- SW00SR is now allocated as contiguous heap storage so that you no longer need to manage chains.
- Core blocks are no longer required to process six or fewer keys in an application program.
- The z/TPFDF product is not supported in the Airline Control System (ALCS) environment.

The following restrictions might require you to make some changes to your application programs.

- You cannot share key lists between C/C++ applications and assembler applications; that is, you cannot set up a key list in a C/C++ application and activate that key list in an assembler application, or vice versa. If you set up a key list in an assembler application program, you must activate that key list in an assembler application program by specifying the KEYLIST or MODLIST parameter on a z/TPFDF assembler macro. If you set up a key list in a C/C++ application program, you must use the dfkey function to activate that key list in a C/C++ application program. See [z/TPFDF Programming Concepts and Reference](#) for more information about setting up and using keys and key lists.
- Assembler applications cannot pass storage addresses above the 2-GB bar to the z/TPFDF product.
- Performance degradation can occur if some parameters in C/C++ applications that reside above the 2-GB bar are passed to the z/TPFDF product. The z/TPFDF product cannot process some parameters above the 2-GB bar; therefore, these parameters are copied below the 2-GB bar and then processed. Copying the data and obtaining any required ECB heap can impact performance. **Table 1** lists the z/TPFDF functions and corresponding parameters that can affect performance. See [z/TPFDF Programming Concepts and Reference](#) for more information about these functions.

**Table 1. z/TPFDF C functions and parameters that can affect performance**

<table>
<thead>
<tr>
<th>z/TPFDF function</th>
<th>Parameters affecting performance if passed above the 2-GB bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>dfadd</td>
<td>• acc parameter when the access parameter is set to DFADD_ALG (pointer to an algorithm)</td>
</tr>
<tr>
<td></td>
<td>• rec parameter (pointer to an LREC)</td>
</tr>
<tr>
<td></td>
<td>• sub parameter (pointer to a subLREC)</td>
</tr>
<tr>
<td></td>
<td>• usr parameter (pointer to a userLREC)</td>
</tr>
</tbody>
</table>
Table 1. z/TPFDF C functions and parameters that can affect performance (continued)

<table>
<thead>
<tr>
<th>z/TPFDF function</th>
<th>Parameters affecting performance if passed above the 2-GB bar</th>
</tr>
</thead>
</table>
| dfadr            | • *alg* parameter (pointer to an algorithm)  
|                  | • *beg* parameter (pointer to an algorithm for the begin ordinal)  
|                  | • *end* parameter (pointer to an algorithm for the end ordinal)  |
| dfcls            | • *alg* parameter (pointer to an algorithm)  
|                  | • *new* parameter (pointer to a new reference name)  |
| dfcpy            | • *acc* parameter when the *access* parameter is set to DFCPY_ALG (pointer to an algorithm) or DFCPY_FADDR8 (pointer to an 8-byte file address)  
|                  | • *toa8* parameter (an 8-byte file address)  |
| dfcre            | *alg* parameter (pointer to an algorithm)  |
| dfdel            | • *acc* parameter when the *access* parameter is set to DFDEL_ALG (pointer to an algorithm)  
|                  | • *lst* parameter (pointer to a list of subfiles)  
|                  | • *nbr* parameter when the *nbr_type* parameter is set to DFDEL_LIST (pointer to a list of LREC numbers)  |
| dfdix            | *alg* parameter (pointer to an algorithm)  |
| dfdsp            | • *acc* parameter when the *access* parameter is set to DFDSP_ALG (pointer to an algorithm)  
|                  | • *acc* parameter when the *access* parameter is set to DFDSP_FADDR8 (pointer to an 8-byte file address)  |
| dfifb            | • *ref* parameter (pointer to a reference name)  
|                  | • *new* parameter (pointer to a new reference name)  |
| dfkey            | *sea* field (search field address) in the key list structure (SW01SR) referenced by the *key_list* parameter  |
| dfmod            | • *acc* parameter when the *access* parameter is set to DFMOD_ALG (pointer to an algorithm)  
|                  | • *sea* field (search field address) in the key list structure (SW01SR) referenced by the *key_list* or *mod_list* parameter  |
| dfmrg            | *key_list* parameter (pointer to a key list)  |
| dfopn            | • *acc* parameter when the *access* parameter is set to DFOPN_ALG (pointer to an algorithm)  
|                  | • *tpn* parameter (pointer to a symbolic tape name)  |
| dfred            | • *acc* parameter when the *access* parameter is set to DFRED_ALG (pointer to an algorithm)  
|                  | • *nbr* parameter when the *nbr_type* parameter is set to DFRED_STACK (pointer to a stack area)  |
| dfrep            | • *rcd* parameter (pointer to a replacement LREC)  
|                  | • *sub* parameter (pointer to a subLREC)  
|                  | • *usr* parameter (pointer to a userLREC)  |
| dfret            | *stk* parameter when the *stk_type* parameter is set to DFRET_STACK (pointer to a stack area)  |
The global area is a portion of fixed main storage that contains application records. The global area permits fast and efficient communication among application programs, and between application programs and the control program. There have been a number of globals changes with the z/TPF system.

In the TPF 4.1 system, the primary global area resides below the 16-MB line and the extended global area resides above the 16-MB line. Because the z/TPF system does not support 24-bit programs, there is no longer a need for the primary global area to be below the 16-MB line. Therefore, the primary and extended global areas have been combined into a single global area that resides below the 2-GB bar. The combination of the primary and extended global areas from the TPF 4.1 system is referred to as format-1 globals in the z/TPF system.

There also is a new global area in the z/TPF system referred to as format-2 globals. Format-2 globals contain many of the characteristics of format-1 globals. However, support for format-2 globals also removes many of the inherent restrictions of format-1 globals and provides improved overall usability. Format-2 globals have the following unique characteristics:

- The format-2 global area contains global records only; there are no global fields in this area.
- Format-2 globals can reside below the 2-GB bar (for 31-bit globals) or above the 2-GB bar (for 64-bit globals).
- There is a separate set of assembler macros and C language functions to access format-2 global records. Format-1 globals use application programming interfaces.

### Additional information:

- See "No longer supported" on page 35 for a list of hardware and software that are no longer supported.
- See "Migrating to the z/TPFDF product" on page 51 for information about specific tasks that you must do.
- See "Coexistence, migration, and fallback" on page 63 for coexistence considerations.
(APIs) that are based on the physical position of the particular global record or
global field in a global directory. Therefore, these APIs depend on the static
nature of global definitions. However, format-2 globals use APIs that are
independent of the physical position of the global record in a global directory. The
layout of the format-2 global directory is transparent to the user application.

- Format-2 globals are defined to the z/TPF system with a new set of commands
  (ZGLBL) that allow you to define and manage format-2 global records
dynamically. There is no offline definition process for these global records.
- There are no size restrictions for format-2 globals.
- There is no maximum number or maximum size for records that can be
  keypointed.
- There is no maximum number or maximum size for records that can be
  synchronized.
- Format-2 globals reside in dynamically allocated areas of system storage.
- You can define format-2 global records to be entirely system-controlled or entirely
  user-controlled. There are new user exits available that allow you to easily
  customize and implement user-controlled globals.
- Updates do not require you to use the system test compiler (STC), pilot tapes, or
  the online data loader.

**Additional information:**
- See [z/TPF Operations](#) for more information about the ZGLBL commands.
- See [z/TPF System Installation Support Reference](#) for more information about
globals and for a list of the user exits that are available.

**Global synchronization**
The SYNCC general macro coordinates the synchronization of format-1 global area
data among several active I-streams and CPCs in a tightly coupled or loosely
coupled environment. When you specify the SYNC parameter on the SYNCC
macro, you also can specify WAIT=YES or WAIT=NO. If you specify WAIT=YES,
the SYNCC macro does not return to the caller until all I-streams have
acknowledged that synchronization is completed.

**Additional information:** See [z/TPF General Services](#) for more information about
the SYNCC macro.

**File system support**
The TPF 4.1 system supported one file system (called the TPF file system). The
z/TPF system supports the following file systems:

- **z/TPF collection support file system (TFS),** previously referred to as the
  TPF file system, which is a subsystem-unique, processor-shared file system that uses
  z/TPF collection support as its storage mechanism and is maintained across
  processor IPLs.
- **Memory file system (MFS),** which is a subsystem-unique, processor-unique file
  system that is not maintained across processor IPLs and is reinitialized every
time it is mounted. MFS uses system heap as its storage mechanism
- **Fixed file system (FFS),** which is a processor-unique file system that is built by
  using a subsystem-unique, processor-shared fixed file record type. That is, only
  one processor can mount the FFS in read/write mode at one time, but multiple
  processors can mount the same FFS in read/only mode. FFS uses fixed file
  records for both i-nodes and data records; therefore, it provides a file system for
  files that need to be maintained across system IPLs where performance is more
important than minimizing the use of file space, and where data must not be exposed to corruption that might be caused by applications inadvertently overwriting pool records.

- **Pool file system (PFS)**, which is a processor-unique file system that is built by using a subsystem-unique, processor-shared fixed file record type for i-nodes and pool records for data. That is, only one processor can mount the PFS in read/write mode at one time, but multiple processors can mount the same PFS in read/only mode. PFS uses fixed file records for i-nodes and pool records for data records; therefore, it provides a file system for files that need to be maintained across system IPLs with a high level of performance, but by using pool records for data, PFS provides a more efficient use of file space.

Additional enhancements for z/TPF file system support include:

- The ZAVFS and ZDVFS commands, which you can use to change and display file system configuration and trace information.
- The ZFILE attr command, which you can use to set, remove, or display attributes of a file.
- The ZFILE fsck command, which you can use to scan the file system for errors and, in some cases, repair those errors.
- The ZFILE mount and ZFILE umount commands, which you can use to mount and umount a file system.
- The ZFILE statfs command, which you can use to display status information about data collection counters or debug information for a specific file system.

**Additional information:**
- See [z/TPF Concepts and Structures](#) and the [z/TPF C/C++ Language Support User’s Guide](#) for more information about the z/TPF file systems.
- See [z/TPF Operations](#) for more information about the file system commands.

**z/TPF application requester (z/TPFAR)**

You still must precompile your z/TPFAR application programs with the DB2 precompiler. However, there are new procedures for doing that precompile because of the z/TPF development environment.

In the TPF 4.1 system, only C language application programs had to be processed through the TPF DB2 postprocessor. With the z/TPF system, both assembler and C language application programs must be processed with the z/TPF DB2 postprocessor.

**Additional information:** See the [z/TPF Application Requester User’s Guide](#) for more information about how to precompile and postprocess your application programs.

**z/TPF collection support (z/TPFCS)**

The ZOODB REUSE command was added to let you manage the z/TPF collection support (z/TPFCS) pool reuse table and several of the ZBROW commands were restructured to make them easier to use.

**Additional information:**
- See [z/TPF Concepts and Structures](#) for more information about z/TPFCS.
- See [z/TPF Operations](#) for more information about the ZOODB REUSE and ZBROW commands.
Data communication

With the z/TPF system you can connect to the network by using SNA PU 5, Advanced Peer-to-Peer Networking (APPN), or TCP/IP native stack support. There are a number of older communications protocols that are no longer supported with the z/TPF system.

Additional information: See “No longer supported” on page 35 for a list of what is no longer supported.

TCP/IP

The z/TPF system includes the following TCP/IP enhancements:

- Large TCP/IP tables were moved above the 2-GB bar, including the socket block table, IP message table (IPMT), and Open Systems Adapter (OSA) read buffers. This allows these tables to become very large.
- TCP/IP messages that are received from the network are read into IPMT blocks instead of 4-KB frames. With this change, you might need to increase the maximum number of IPMT entries that are defined for your system.
- All socket APIs support 64-bit addresses.
- New socket traces are available. One is at the socket level and the other is at the ECB level.
- You can now poll OSA connections in 1052 state.
- You can create and use sockets in 1052 state, and start the Internet daemon in 1052 state.
- You can set OSA input message priority on a per-application or per-socket basis; high-priority messages can be read and processed even when in input list shutdown.
- The z/TPF system reads input messages from OSA connections during dump processing to reduce the likelihood of lost messages during system errors. The input messages are queued and processed after the dump is completed.
- You can use the ZSOCK command to display active sockets that have the most exceptions or the most exceptions of a specific type (such as retransmits, data out of order, and so on).
- The IP scan process can run on any I-stream in the z/TPF system. In the TPF 4.1 system, this was limited to only the main I-stream.

Additional information:

- See z/TPF Transmission Control Protocol/Internet Protocol for more information about TCP/IP support, including information about tuning major control block structures.
- See z/TPF Operations for more information about the ZSOCK command and other TCP/IP commands.

Application programming

The following information describes some of the changes related to application programming for the z/TPF system and the z/TPFDF product:

- Large memory (64-bit) is available to applications; not just to the system.
- You now can use register 8 (R8), R10, R11, R12, and R13 in your applications. In the TPF 4.1 system, the contents of R10-R13 were not guaranteed across a macro call. However, the z/TPF system provides an extended register save function that allows you to choose whether the contents of R10-R13 will be
saved across a macro call. To use the extended register save function, specify the EREGSAVE parameter on the BEGIN or DEFBC general macro.

- You can use a register other than R8 for the base register, and you can have multiple base registers or no base register. Use the BEGIN macro to define the base register or to specify that you do not want a base register.

- Timeout processing was enhanced in the following ways:
  - In the TPF 4.1 system, there was one system-wide timeout value and the only way to set or change the timeout value was by calling the SETOC system macro. In the z/TPF system, there is a timeout value defined for each program in the tpf.cntl file and you can use the ZAPAT command with the TIMEOUT parameter specified to change the timeout value for each program online.
  - In the TPF 4.1 system, timeout processing did not run until the system was above 1052 state; in the z/TPF system, this is not true. Therefore, you must update the program attribute table (PAT) for any application programs that have prolonged looping logic and run in restart or 1052 state to prevent a timeout error.
- Previously, transfer vectors were specified in IBMPAL, but with the changes to the PAT for the z/TPF system, you now must code transfer vectors on the BEGIN macro.

Because transfer vector entries are no longer included in the PAT, the PROGC system macro and progc function were updated as follows:

- You no longer can specify PARENT=NO on the PROGC macro. However, to support single source applications, if your applications specify PARENT=YES on the PROGC macro, you do not have to change that coding. When the application program runs on the z/TPF system, the PARENT=YES specification will be ignored.
- The PAT_PBI and PAT_DBI values are no longer supported for the progc function. You must change any applications that specify these values.

- The LODIC general macro now supports 4 more user classes. In addition, you can specify the maximum amount of time that the ECB will give up control because of low system resources.

Additional information:

- See z/TPF General Services or z/TPF System Services for more information about register conventions.
- See z/TPF General Services for more information about the BEGIN, DEFRC, and LODIC macros.
- See z/TPF Operations for more information about the ZAPAT command.
- See z/TPF System Services for more information about the PROGC macro.
- See the z/TPF C/C++ Language Support User's Guide for more information about the progc function.

With the move to 64-bit architecture and to the GNU compiler collection (GCC), there are some changes that you must make to your application programs. The following describes some general considerations for C language and assembler programs. See "Migrating your application programs" on page 57 for more information about the specific changes required.

C language programs

The following information describes some of the changes that are specific to C language programs:
• Header file changes. The GCC does not allow header file names with $ characters.

• Differences between the IBM C/C++ compiler and the GCC. The GCC does not support trigraphs and expects the backslash (\) character to be the very last character on the line.

• 64-bit considerations:
  – With the z/TPF system, all C/C++ programs are compiled as 64 bit, which defines the size of a long data type as 8 bytes. With the TPF 4.1 system, all C/C++ programs are compiled as 32-bit, which defines the size of a long data type as 4 bytes. For both systems, the size of an integer data type is 4 bytes.
    To provide compatibility with the size of the fields contained in data structures that have corresponding assembler data areas (DSECTs), all long data types were changed to integer data types. As code was converted to 64 bit, selected fields were changed back to long data types as appropriate.
    You need to decide if the data type size change affects your applications and how you would like to make your changes. The convert_long_int.sh conversion tool can help you make these changes.
  – Another 64-bit consideration is the use of pointers. When a C/C++ program is compiled as 64-bit, the size of a pointer data type is 8 bytes. Sometimes pointers are cast to integers for various reasons. To create code that is compatible for both 32-bit and 64-bit systems, change the casting from integer to long. The GCC produces a warning for pointers that are cast to integers.
    There might be some instances where you will continue to need a 4-byte pointer; for example, pointers to core blocks in the ECB cannot be increased to 8 bytes. To provide compatibility, a 31-bit pointer data type called __ptr32_t and a 31-bit pointer attribute called PTR32ATT are provided.
  – When a C/C++ program passes a pointer to static or constant data to an assembler program, the called assembler program must handle the pointer as a true 64-bit pointer.

• With the TPF 4.1 system, the standard template library is supported by STLPort. With the z/TPF system, the standard template library is supported by the GNU standard C++ library (libstdc++).

• The TPF 4.1 system supports floating point data only in hexadecimal floating point (HFP) format. However, the GCC (and the z/TPF system) supports only binary floating point (BFP) format. You can use the following functions to migrate your floating point data, and to maintain single source applications that run under both the TPF 4.1 system and the z/TPF system:
  – tpf__fp_htob
  – tpf__fp_btoh
  – tpf__fp_hton
  – tpf__fp_ntoh
  – tpf__fp_bton
  – tpf__fp_ntob.

Additional information:
• See “Mapping data types” on page 81 for more information about the difference in data types.
• See the z/TPF C/C++ Language Support User’s Guide for more information about the floating point functions.
• See “Migrating your application programs” on page 57 for more information about specific changes required for your application programs.
**Assembler programs**

The following information describes some of the changes that are specific to assembler programs:

- In the TPF 4.1 system, assembler programs were limited to 4 KB in size; in the z/TPF system, assembler programs can be larger than 4 KB. To exploit this capability, you can change your assembler programs to use:
  - The CLINKC, RLINKC, and SLINKC assembler linkage macros
  - Multiple base registers
  - Baseless instructions.

- You can use the CALLC general macro in assembler programs to call C language functions.

- In the TPF 4.1 system, the TMSPC and TMSEC macros were provided to set up the interface between C language programs and macro service routines written in assembler language. In the z/TPF system, the PRLGC and EPLGC macros set up this interface by simulating the prolog and epilog code generated by the GCC. The PRLGC and EPLGC macros were provided on the TPF 4.1 system through APAR PJ29640 so that new C library functions written on the TPF 4.1 system can be migrated with little or no changes; and the TMSPC and TMSEC macros are still supported on the z/TPF system so that library functions that were already coded with those macros can be migrated with little or no code changes. New library functions that are developed for z/TPF system must be coded with the PRLGC and EPLGC macros.

**Note:** Some TPF 4.1 library functions will require changes depending on how they are used. See "Migrating your C library functions" on page 55 for details about the changes that might be required.

- z/TPF basic assembler language (BAL) programs can be repackaged into multiple entry point shared object libraries to provide the following performance advantages and benefits:
  - The program nesting level (PNL) has been replaced with the application stack. Maintaining the PNL information in the application stack is more efficient than keeping it in a separate control structure.
  - An application stack frame is created for each BAL segment, which allows the application stack to be used by z/TPF assembler applications.
  - Reduction of system overhead because of better program linkage (Enter/Back processing). Linkage has been changed from a system service to a function call (similar to C language).
  - Reduction in system overhead for an application because two segments can be linked together as one unit, creating a shorter path length.
  - Program expansion beyond 4 KB.

- There are no longer macros that begin with a dollar sign ($); some of these macros were removed completely, and others were replaced by or merged with a corresponding non-$ macro. The following are some additional considerations related to the $ macros:
  - The $DISBC, $GCOMC, $GETBC, and $GSWBC system macros were replaced by the DISBC, GCOMC, GETBC, and GSWBC system macros. In addition, these macros now have a required OWNER parameter for block owner support. See "Main storage" on page 10 for more information about block owner support.
  - The $MONTC system macro, which provided an optional OLDSTATE parameter to save the first half (4 bytes) of the current PSW, is no longer
supported with the z/TPF system. The MONTC system macro provides the same ability to save the first half (8 bytes) of the current PSW with the new PREVSTATE parameter.

- There are more assembler instructions available for your use in application programs. In addition to the z/Architecture instructions for 64-bit operations (such as LG and STG), the z/TPF system now supports the immediate set of instructions (such as AHI, CHI, and so on).

Additional information:
- See z/TPF General Services for more information about CALLC macro.
- See z/TPF Application Programming for more information about BAL repackaging, the application stack, and coding your own library functions.
- See z/TPF System Services for more information about the DISBC, GCOMC, GETBC, GSWBC, and MONTC macros.
- See “Migrating your C library functions” on page 55 for more information about specific changes that are required to migrate your own C library functions.
- See “Migrating your application programs” on page 57 for more information about specific changes that are required for your application programs.

User Exits

The z/TPF system and z/TPFDF product include several new user exits to help you customize your systems. Some user exits that exist in the TPF 4.1 system are no longer supported or have been replaced with new user exits in the z/TPF system.

In the z/TPF system, when control is passed to a control program (CP) user exit for processing, it is done in 64-bit addressing mode; therefore, any modifications that you have in the control program must be converted to use 64-bit addressing mode. Also, the user exit override address field has been expanded to 8 bytes. If you use this address or use hard coded displacements when you build a user exit parameter list, you must change the code that builds the list.

Additional information:
- See z/TPF System Installation Support Reference for a list of user exits that are new, changed, or no longer supported. Details about each user exit are provided in the user exit code commentary.
- See “Customizing the code” on page 53 for more information about specific changes that you might need to make for user exits.

Build environment

With the z/TPF system and z/TPFDF product, the build environment is moving to a Linux-based environment. The MakeTPF build solution is a complete set of tools that are used for assembling, compiling, and linking z/TPF system, z/TPFDF product, and customer application programs. The MakeTPF build solution consists of two stages:

- The system initialization process (SIP), where you define the environment for your z/TPF system and z/TPFDF product.
- Build, which actually creates (or builds) the components to be loaded to the system.

The MakeTPF build solution requires a cross compiler to build application and system programs that are written in C/C++ language. For the z/TPF system, you
must build the GNU compiler collection (GCC) as a cross compiler on Linux; that is the compiler is built and used on Linux, but the compiler produces code that runs correctly on the z/TPF operating system.

Additional information:
- See “Operating environment requirements and planning information” on page 69 for information about software requirements for building programs and generating the z/TPF system and z/TPFDF product.
- See “Generating the z/TPF system” on page 45 for information about changes you need to make to build programs and generate the z/TPF system and z/TPFDF product.
- See z/TPF and z/TPFDF System Generation for more information about SIP and generating the z/TPF system and z/TPFDF product.
- See z/TPF Program Management for more information about the MakeTPF build solution and building the cross compiler.

Middleware and Web services

The following information describes changes in middleware and Web services for the z/TPF system.

WebSphere MQ for z/TPF

WebSphere MQ for z/TPF (previously referred to as MQSeries) takes advantage of a number of the new functions that are provided in the z/TPF system:
- Dumps
- Named blocks
- Trace groups
- High-priority TCP/IP messages.

Some ZMQSC commands also were enhanced so that they can be processed in 1052 state if pools are active.

Additional information:
- See z/TPF Application Programming for more information about WebSphere MQ for z/TPF.
- See z/TPF Operations for more information about the ZMQSC commands.

Features

The z/TPF system includes the base product and the High Performance Option (HPO) feature. This feature, consisting of the loosely coupled facility and the multiple database function (MDBF), allows the z/TPF system to run in a loosely coupled configuration where each central processing complex (CPC) can share a common database.

The multi-processor interconnect facility (MPIF) and z/TPF application requester (z/TPFAR), which were provided as separate features for the TPF 4.1 system, are included in the z/TPF base product.

Additional information:
- See z/TPF Main Supervisor Reference for more information about the HPO feature.
Problem determination

The following information describes new and changed problem determination tools that are available with the z/TPF system.

Binary utilities

The GNU compiler collection (GCC) includes a set of binary utilities called binutils. The binary utilities include tools such as readelf and objdump, which you can use to view program information on Linux while doing problem determination. For more information about GCC, go to http://www.gnu.org.

System error dumps

See “System error processing” on page 14 for information about enhancements that were made to dumps; also see z/TPF Program Development Support Reference for information about new, changed, and obsolete dump tags.

Trace support

Trace support in the z/TPF system has been updated to make traces more flexible and easier to use. The following information describes some of the key enhancements:

- The macro trace and C function trace facilities that were provided with the TPF 4.1 system have been combined; this single trace facility is now referred to as ECB trace. Use the ZSTRC command to start and stop the ECB trace.
- ECB heap trace provides tracing capabilities in addition to the ECB trace. The ECB heap trace provides information about the latest ECB heap storage requests and releases for an ECB.
- Socket trace allows you to trace socket APIs and debug TCP/IP socket applications. Two types of socket trace are available:
  - The socket level trace traces all socket APIs that were issued on a given socket. When the socket API trace is enabled, you can display the socket level trace online using the ZSOCK command.
  - The ECB level trace traces all socket APIs that were issued by a given ECB. When the socket API trace is enabled, the ECB level trace is included and formatted in dump output.
- You can now turn the register trace facility on or off without IPLing the system.
- Trace groups define how application trace data is separated from system trace data to maintain as much information as possible. Use trace group levels to map trace data to different buffers for a given ECB.
- The trace log facility allows you to trace application function and macro calls to a file or real-time tape so that application debugging is not disruptive; that is, without using dumps or the z/TPF debugger.
- The I/O trace facility was enhanced to include the ZIOTR DISPLAY and ZIOTR SET COUNT commands. The ZIOTR DISPLAY command lets you display I/O trace information online. The ZIOTR SET COUNT command lets you change the number of trace entries that are used to record trace information for one or more I/O devices.
Heap check mode

Heap check mode, which is similar to the existing block check mode, is a debugging tool that flags certain coding errors related to the use of ECB heap to help you determine when an application is changing or accessing storage beyond its allocated heap buffer or accessing an ECB heap buffer after it was released. See z/TPF Main Supervisor Reference for more information about heap check mode.

Physical block owner support

With the z/TPF system, the OWNER parameter on selected macros allows you to identify which z/TPF system component is actually using the storage. The following macros allow you to use the OWNER parameter:

- DISBC
- GCOMC
- GETBC
- GLFMC
- GSWBC.

Core blocks are tagged with the system component (for example, ITAPE, IDASD, and so on) that obtained the block. If the system is running low on core blocks, you can enter the ZSTAT command with the OWNER parameter specified to display the block usage and determine which component is having a problem.

z/TPF debugger

With the z/TPF debugger, you can now view a captured ECB dump on the z/TPF console. You also can take a snapshot of a long-running ECB and view it with the debugger. See z/TPF Application Programming for more information about the z/TPF debugger and how to use it.

Note: The z/TPF debugger only supports the IBM TPF Toolkit for WebSphere Studio.

Stand-alone dump and postprocessor

The stand-alone dump (SADUMP) utility is an offline utility that you can use to dump various storage areas when a normal online dump cannot be processed because of a system hang condition. For the z/TPF system, the SADUMP utility was updated to handle z/Architecture machines. The areas included in the dump also have been updated.

The stand-alone dump postprocessor (PPSADUMP) utility is a C language program that runs on Linux to create a printed storage dump from the data that was
captured on the SADUMP tape. The PPSADUMP utility replaces the SADPRT utility on the TPF 4.1 system that ran under IBM MVS control.

**Additional information:**
- See [z/TPF Operations](#) for more information about the SADUMP and PPSADUMP utilities.
- Enter `man ppsadump` from your Linux system for more information about the PPSADUMP utility.

### Performance

The following information describes changes in the tools and utilities that are available for tuning the performance of your z/TPF system.

#### Data collection and reduction

A number of changes were made to the data collection and reduction reports to support the z/TPF system. See [z/TPF System Performance and Measurement Reference](#) for more information about and sample reports from data collection and reduction.

#### Continuous data collection

The continuous data collection (CDC) performance tool is a collector/server application and client application that combine to provide a real-time z/TPF resource monitor. Vital system metrics are displayed that give you an overview of system state and performance. You can use the display as an early warning system, use the information gathered over time for future z/TPF capacity planning, or use several hours worth of information to help solve performance problems.

CDC was enhanced significantly with TPF 4.1 program update tape (PUT) 19 (APAR PJ29925). If you are migrating to the z/TPF system from a TPF 4.1 system that is at an earlier PUT level, and if you have written analysis programs or other postprocessing code for the previous version of CDC, you must change those programs to handle the new data formats.

**Additional information:**
- See [z/TPF System Performance and Measurement Reference](#) for more information about CDC.
- See “Coexistence, migration, and fallback” on page 63 for coexistence considerations.

#### z/TPF software profiler

The z/TPF software profiler is a tool that helps you to understand the performance characteristics of your applications. This performance measurement tool collects and analyzes data for:
- External interrupt (EI) analysis, which is a sampling tool that identifies the instructions and applications that are processed the most.
- ECB entry analysis (EA), which shows all macros and functions that are processed by a specific ECB.
- Macro analysis (MA), which identifies the supervisor call (SVC) macros that are processed the most and by which applications.
- Page range (PR) analysis, which identifies if 4 KB pages in memory are being accessed.
Resource consumption (RC), which collects ECB specific performance data.

The z/TPF software profiler consists of two components:

- An online z/TPF software profiler command called ZTRAP, which collects performance data by taking samples at timed intervals and writing the data to tape for offline processing.
- A z/TPF software profiler offline program that selects data runs from the tape and reduces, formats, and reports the data based on the control option information that you specify to the offline program.

The offline program and changes to the online control program (CP) are available for download from [http://www.ibm.com/tpf/download/tools.htm](http://www.ibm.com/tpf/download/tools.htm).

Additional information:
- See [z/TPF Operations](http://www.ibm.com/tpf) for more information about the ZTRAP command and the types of data collection runs that are supported.
- See [z/TPF System Performance and Measurement Reference](http://www.ibm.com/tpf) for more information about using the offline analysis program.

### Operations

The following information describes some miscellaneous command changes for the z/TPF system. See [z/TPF Operations](http://www.ibm.com/tpf) for details about all new, changed, or deleted commands.

- The ZACNF and ZDCNF commands were added to allow you to change and display data by referencing the CINFC label.
- The ZACOR and ZDCOR commands were changed to support 8-byte addresses. Also, if the address specified on the ZDCOR command is not on a fullword boundary, no fullword boundary alignment is done; that is, the display begins at the address that you specify.
- The LOCK parameter is no longer supported on the ZRPGM command. Use the ZRPGM command to retrieve a program into memory and display the address.
- The ZDMAP command displays link map data for all programs (including the control program), both assembler and C language. You can use the ZDMAP command to find the start of the executable program.
- The ZAPAT command has a new parameter (ADD) that allows you to add a program to the program attribute table (PAT).
- The ZDECB command has a new parameter (TR) to display an ECB trace for an ECB that is in use. This parameter replaces the MTR parameter.
- The ZDECK command was added to let you extract data from an online z/TPF table to a hierarchical file system (HFS) file that can be used as input to an offline program to keep offline information synchronized with online information.
- The ZDSVC command was updated with a new parameter (CODE) that lets you display a macro name by specifying the SVC number.
- The ZAPFS and ZDPFS commands were added to manage positive feedback support, which provides a way to keep offline information for your z/TPF system synchronized with your online information. See [z/TPF Database User’s Guide](http://www.ibm.com/tpf) for more information about positive feedback support.
- The ZDSYS command has a new parameter (ALL) that lets you display the system state for all subsystems.
- The ZRTDM MODIFY command was updated to allow you to add or change a record ID online; you do not need to load a RIAT.
The disassembled view that is available with the ZDCOR and ZDPGM
commands and the z/TPF debugger was enhanced to support the z/Architecture
instruction set. In addition, z/TPF macro parameters now are included in the
disassembled view.

Migration tools

The following information describes tools that are available to help you with your
migration.

Code migration tools

Table 2 shows the various code migration tools that are available to you from
http://www.ibm.com/tpf/download/tools.htm Also, see "Migrating your application
programs" on page 57 for more information about when and how to use some of
these tools.

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>convert_hdr.sh</td>
<td>Converts all $ characters in header file names to underscore (_) characters. The names of the files and the names of other header files contained in the file are updated. This tool processes files with extensions of .h, .hpp, .include, and .inl.</td>
<td>See the convert_hdr.README file for more information.</td>
</tr>
<tr>
<td>convert_src.sh</td>
<td>Converts all $ characters in header file names to underscore (_) characters. The names of the files and the names of other header files contained in the file are updated. This tool processes files with extensions of .c, .cpp, .asm, .l, .sqlc, .y, and .bsc.</td>
<td>See the convert_src.README file for more information.</td>
</tr>
<tr>
<td>add_tpf_hdr.sh</td>
<td>Adds /tpf to the name of any unique z/TPF header file referenced in the file. This tool processes files with extensions of .h, .hpp, .include, .inl, .c, .cpp, .asm, .l, .sqlc, .y, and .bsc.</td>
<td>Run the tools that convert the $ characters before you run this tool. See the add_tpf_hdr.README file for more information.</td>
</tr>
<tr>
<td>rmtrigraphs</td>
<td>Removes trigraph statements that are related to file tags and compiler version.</td>
<td>GCC does not support trigraphs. See the rmtrigraphs.README file for more information.</td>
</tr>
<tr>
<td>rmbackslashes</td>
<td>Removes trailing blanks after the backslash characters in macros.</td>
<td>GCC expects the backslash character () to be the very last character on the line. See the rmbackslashes.README file for more information.</td>
</tr>
<tr>
<td>convert_long_int.sh</td>
<td>Converts long data types to integer (int) data types.</td>
<td>This tool is not required for migration, but is useful if the data type size changes will affect your applications. See the convert_long_int.README file for more information.</td>
</tr>
<tr>
<td>cvtto64</td>
<td>A 64-bit assembler conversion aid that:</td>
<td>See the cvtto64.README file for more information about these tools and how to install and use them.</td>
</tr>
<tr>
<td></td>
<td>• Converts register instructions for use in 64-bit mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Converts 31- and 32-bit values to 64-bit register values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Converts most branch instructions to branch relatives</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Converts applicable literal instructions to immediate instructions, where appropriate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Converts macro names as needed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Converts obsolete structured programming macro (SPM) names to the appropriate replacement macro name.</td>
<td></td>
</tr>
</tbody>
</table>
Table 2. Code migration tools (continued)

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sspragmas</td>
<td>Converts #pragma statements and _Packed directives so that they are compatible with both the TPF 4.1 system and the z/TPF system.</td>
<td>See the sspragmas.README file for more information.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> There are additional changes required to complete the migration for the _Packed directives. See <a href="#">&quot;Migrating your application programs&quot; on page 57</a> for more information about these changes.</td>
<td></td>
</tr>
<tr>
<td>maketpf.bsc.convert</td>
<td>Converts TPF 4.1 build scripts to z/TPF makefiles.</td>
<td>Enter <code>man maketpf.bsc.convert</code> for information about how to use this tool.</td>
</tr>
<tr>
<td>convert_usrtpf2cntl</td>
<td>Converts all entries in a usrtpf.cntl file into the control file format.</td>
<td>This script is a template; you must update this for your specific conventions. See the convert_usrtpf2cntl.README file for more information.</td>
</tr>
</tbody>
</table>

### Information center migration search

The product information for each command, macro, C function, and message includes a field with the heading **Last updated** followed by an indication about whether the item was added, changed, or obsoleted for a particular program update tape (PUT). For the initial release, the indication is PUT00.

To determine what interfaces have been added, changed, or deleted for the initial version of the z/TPF system and the z/TPFDF product, search the **TPF Product Information Center** for the phrase “PUT00”.

See the information center help ([TPF Product Information Center Home->Help](#)) for more information about searching in the information center.

#### Important note

Search results for the information center are limited to 500; therefore, do the following to ensure that your search results are complete:

1. Use the **Migration Search** topic for your search instead of the main product topic.
2. Limit the scope of your search to a specific navigation topic in the migration search topic. See the information center help for information about how to limit the scope of your search.
3. Place the search term or phrase inside quotation marks (for example, “PUT00”) to ensure that the search engine searches for that exact term or phrase. If you do not use the quotation marks, the search engine will look for any term with the string PUT.

You also can print the results directly or save the results to a file for future reference. See the information center frequently asked questions ([TPF Product Information Center Home->Help->FAQs](#)) for more information about how to print or save your results.

### No longer supported

The following lists the hardware and software that are no longer supported with the z/TPF system and z/TPFDF product. See ["Operating environment requirements and planning information" on page 69](#) for information about the hardware and software requirements for the z/TPF system and z/TPFDF product.
Hardware

The following devices are no longer supported:
- IBM S/390 Parallel Transaction Servers (9672 E/P Models)
- IBM S/390 Parallel Enterprise Servers (9672 R Models)
- IBM S/390 Multiprise 2000 Server
- IBM Direct Access Storage Facility (Models 3350, 3375, 3380\(^1\), 3390\(^1\), 9345)
- IBM RAMAC Array DASD
- IBM Storage Control Units (Models 3880, 3990-2, 3990-3, 3990-6)\(^1\)
- IBM Model 3480 Tape Unit and 3495 Tape Library
- IBM Communication Controllers (Models 3702 and 3725)
- IBM Terminal Control Unit (Models 3172 and 3174)
- IBM 3088 Multisystem Channel Communication Unit
- IBM 3505 Card Reader\(^1\)
- IBM 1403 Line Printer\(^1\)
- Limited Lock Facility (LLF) and Extended Limited Lock Facility (ELLF)
- Sysplex Timer (STR) subset
- Time-of-Day (TOD) Clock RPQ

Software

The following software is no longer supported:
- 24-bit addressing mode
- 24-bit global support
- File resident programs
- Format-0 channel command words (CCWs)
- Keypoint fallback extents
- Program test vehicle (PTV)
- Real-time trace (RTT)
- TARGET(TPF)
- VEQR mode
- SLC and bisync communications protocols
- CLAW offload support for TCP/IP
- PU 2.1 logon manager
- Multiple assemble/compilation program (MASM)
- Multiple assembly/compilation print program (ASMP)
- Variable cross reference (VCRS)
- Macro cross reference (DCRS)
- TPFDF Distributed Data Access (DDA) feature.

\(1\). While this device is no longer supported, the device format is still supported.
Part 2. Tasks
Planning for your migration

The following checklist can help you plan and implement your migration from a TPF 4.1 system to a z/TPF system and from the TPFDF 1.1.3 product to the z/TPFDF product.

You might want to plan for the migration by installing and stabilizing the z/TPF system without incorporating the new functions that are provided. Installing a new release without initially exploiting functions allows you to create a stable z/TPF environment.

The items in this checklist are intended to be used in sequential order for a typical z/TPF system. Use this checklist while planning and implementing your migration. Place a check mark (✓) beside each item as you complete it.

1. Obtain commitment and dedication to the migration plan:
   - Review notes from previous migration plans (if any) and identify the need for a new migration plan.
   - Identify resources. These resources include a project leader and personnel who can represent key areas throughout the life of the project. The following is a list of recommended areas for representation.
     - System internals
     - System database
     - System tools
     - Communications
     - Automation
     - Operations
     - Coverage
     - Applications
     - Test systems
     - Linux
     - z/VM
     - IBM representative
     - Other vendor representatives
     - Third party software.

2. Prepare and document a migration plan including the following key topics:
   - Identify hardware, software, and tools (IBM and non-IBM) needs for the z/TPF system and z/TPFDF product:
     - Document your existing environment (list hardware and software tools that you now have)
     - Identify hardware and software that was supported with the previous version, but is no longer supported.
     - Identify new hardware and software requirements
     - Identify changes that are needed for your build processes
     - Identify changes that are needed for your library system (also referred to as SCM or VCS)
- Identify network requirements.
- Identify new or changed data set requirements
- Identify main storage requirements

See "Operating environment requirements and planning information" on page 69 for more information about the hardware and software requirements for the z/TPF system and z/TPFDF product; also see "Preparing your environment" on page 43 for information about tasks needed to prepare your environment. These tasks might help you to identify some of these needs.

- Identify new records, changes required to existing records, and records that are no longer supported to ensure that you have enough DASD for the z/TPF system. See "Generating the z/TPF system" on page 45 for information about record changes. See z/TPF and z/TPFDF System Generation for information about defining records for the z/TPF system and z/TPFDF product.

- Identify naming conventions. See z/TPF and z/TPFDF Programming Standards for more information about naming conventions.

- Identify the migration aids that are available. See "Migration tools" on page 34 for more information.

- Identify training and education requirements. The following personnel will need various levels of training:
  - Application programmers
  - Automation personnel
  - Coverage programmers
  - Operators
  - Planning personnel
  - System programmers
  - System support personnel, such as z/VM, Linux, z/OS, implementation control, library management support, and so on.

- Identify changes to interfaces; such as:
  - Identify changes to commands and messages that will affect your automation procedures.
  - Identify changes to macros and C or C++ functions that will affect your system and applications programs.

**Note:** To determine what interfaces have been added, changed, or deleted for the z/TPF system and the z/TPFDF product, search the TPF Product Information Center for the phrase “PUT00”. Use the Migration Search topic to ensure that your results are complete. See "Information center migration search" on page 35 for more information about searching for migration information in the information center.

- Estimate the capacity impact of the z/TPF system; for example, central processing unit (CPU) utilization for a message, storage requirements, DASD input/output (I/O) access, network controller, and link requirements.

- Identify changes that are required for application programs; for example, all C programs must be converted to be compatible with the GNU compiler collection (GCC). See "Application programming" on page 24, "Migrating your C library functions" on page 55, and "Migrating your application programs" on page 57 for more information about application
program changes; also see "Single source and migration APARs" on page 77 for information about single source APARs to apply to your TPF 4.1 system and TPFDF 1.1.3 product.

- Evaluate customized code (user modifications):
  - Evaluate your existing modifications to determine whether you need to carry any of the modifications forward; for example, new functions available with the z/TPF system and z/TPFDF product might replace some existing modifications. If you do need to carry forward any of the modifications, ensure that you review that code.
  - Evaluate new, changed, and no longer supported user exits. Determine whether programs that are using these exits need to be changed and whether there are user exit alternatives to user modifications.
  - Evaluate changes to interfaces (commands, messages, macros, and C functions) that can affect code that you have written. See "Customizing the code" on page 53 for more information about customizing the z/TPF and z/TPFDF code.

- Identify the migration tasks that you can do to the TPF 4.1 system and TPFDF 1.1.3 product before receiving the z/TPF system and z/TPFDF product. See "Preparing your environment" on page 43 for information about preparing your system for migration.

3. Develop a test strategy that includes the following considerations:
   - Test high-exposure operations.
   - Test tools (IBM and non-IBM).
   - Test application programs.
   - Test using teleprocessing network simulator (TPNS).
   - Test offline.
   - Test operations, including any automation packages that you may have installed. Use the online operations staff to test before the migration.
   - Test the utilities (capture and restore, recoup, online load, data collection, and so on).
   - Perform a system test.
   - Perform a regression test.
   - Test communications.
   - Test the inter-node communications.
   - Test the network.
   - Test any of your modifications.
   - Determine the impact of testing ongoing migration plans.
   - Determine the potential impact on the IBM z/VM system.

**Note:** You can use memory configurations to set up one keypoint A (CTKA) to have configurations for both production and test systems.

4. Develop an education strategy that takes into consideration:
   - Offerings available from IBM
   - Changes to internal education.

5. Develop a fallback plan.

6. Prepare your environment. See "Preparing your environment" on page 43 for more information.
7. Install and start the z/TPF system and the z/TPFDF product. See “Generating the z/TPF system” on page 45 and “Migrating to the z/TPFDF product” on page 51 for more information. Also, if you plan to run a TPF 4.1 system and z/TPF system in the same loosely coupled complex, see “Coexistence, migration, and fallback” on page 63 for coexistence considerations.

8. Implement any changes required for your network.

9. Implement any changes to operational procedures.

10. Migrate your application programs:
   - Convert all C/C++ application programs to be compatible with the GCC.
   - Apply single source APARs and make changes to your application programs as appropriate. See “Single source and migration APARs” on page 77 for more information.
   - Determine which modifications (for example, macros, interfaces, user exits, C functions, and so on) are required for existing application programs.

See “Migrating your application programs” on page 57 for more information.

11. Diagnose and debug problems:
   - Review the new and changed dump formats, trace facilities, and debugging considerations.
   - Review the new and changed ways to record program errors and to use new tools.

See “Problem determination” on page 30 for more information about these changes.

12. Exploit new z/TPF system and z/TPFDF product functions.
Preparing your environment

Complete the following steps to get your environment and system ready for migration.

__ 1.   Ensure that your TPF 4.1 system and TPFDF 1.1.3 product are at the correct maintenance level. See "Maintenance upgrade requirements" for more information.

__ 2.   Migrate your library system to one that supports a hierarchical file system (HFS). See "Operating environment requirements and planning information" on page 69 for more information about available products.

__ 3.   Set up a Linux development infrastructure, including gcc, binutils, and gnumake. See the information provided with the code distribution; also see "Software (programming requirements)" on page 71 for information about where to get Linux for zSeries.

__ 4.   Unpack the z/TPF and z/TPFDF source code into the root directory of an HFS structure. See the readme file on the product CD-ROM for more information; details about unpacking are not included here.


__ 6.   Go to [http://tpf.ncsa.uiuc.edu](http://tpf.ncsa.uiuc.edu) and download the z/TPF modified versions of the glibc and libstdc++ libraries.

__ 7.   Build the GNU compiler collection (GCC) for the z/TPF system. See z/TPF Program Management for more information.

Maintenance upgrade requirements

For the TPF 4.1 system, complete the following steps:

__ 1.   Upgrade the TPF 4.1 system to PUT 15 or later.

__ 2.   Apply single source and migration APARs to your TPF 4.1 system as appropriate. See "Single source and migration APARs" on page 77 for more information.

Notes:

a.   If you plan to run a TPF 4.1 system and z/TPF system in the same complex, apply PUT 18 APAR PJ29083.

b.   If your TPF 4.1 system uses prime/duplicate module pairing, apply PUT 19 APAR PJ29358.

c.   If you use the TPF Internet mail server, apply PUT 19 APAR PJ29906.

d.   Apply APAR PJ30220.

__ 3.   Perform the 32-way loosely coupled conversion:

a.   Apply 32-way loosely coupled processor support (APAR PJ27785).

b.   Enter the ZMIGR command to migrate to 32-way loosely coupled processor support.

Note: You must do this step even if your system is not loosely coupled.

__ 4.   Perform the FARF6 conversion:

a.   Apply 8-byte file address support (APAR PJ28097).

b.   Enter ZMODE 6 to activate FARF6 8-byte file addressing.
Note: You are not required to use FARF6 addressing; this support must be started for z/TPF system use.

If you currently have the TPFDF 1.1.3 product, complete the following steps:

1. Update the TPFDF 1.1.3 product to PUT 16 or later.
2. Apply the following corequisite APARs from TPFDF PUT 17:
   - PQ49400 – Compatibility with 8-byte pool file addresses
   - PQ50700 – 32-way loosely coupled support.
3. Apply single source and migration APARs to your TPFDF 1.1.3 product as appropriate. See "Single source and migration APARs" on page 77 for more information.
Generating the z/TPF system

Complete the following steps to install and build a z/TPF system.

__ 1. Modify the system initialization process (SIP) input deck (previously called the SIP stage I deck). See "Modify your SIP input deck" for more information.

__ 2. Make the following changes to your system communication keypoint (SCK) macros:
   - Update the DEVICE parameter on the SENDG macro if you specify any devices that are no longer supported.
   - Delete the PKSTG macro.
   See z/TPF and z/TPFDF System Generation for more information about the SCK macros.

__ 3. Migrate the usrtpf.cpy file to the control file called usr.cntl. Use the convert_usrtpf2cntl tool to do this. See the readme file for more information about how to use this tool.

   **Note:** In the z/TPF system, control file tpf.cnt1 replaces IBMPAL. If you made changes to IBMPAL, you must make the appropriate changes to the tpf.cnt1 file.

__ 4. See z/TPF and z/TPFDF System Generation for information about how to generate and build the z/TPF system; and see "Coexistence, migration, and fallback" on page 63 for information about other considerations that you must be aware of.

---

Modify your SIP input deck

__ 1. Remove all SIP macros that are no longer supported from your SIP input deck, and review changes to all SIP macros and skeletons. See "Changes to SIP macros" for more information.

__ 2. Update your RAMFIL statements. See "Changes for the RAMFIL statements" on page 47 for more information about changes needed for records.

__ 3. Generate a new FACE table using the FACE table generator (FCTBG) program. See z/TPF and z/TPFDF System Generation for information about how to generate the FACE table.

---

Changes to SIP macros

The following list includes some of the changes that you need to make to your SIP macros. See z/TPF and z/TPFDF System Generation for more information about the SIP macros. You also can get a list of all macros that were changed by searching for the string “PUT00” in the information center. See "Information center migration search" on page 35 for more information about searching for migration information in the information center.

- Delete the BSCER and SLCER parameters from the CCPERR macro.
- Delete the AIDWN, AIRES, BSDWN, and BSRES parameters from the CCPPOL macro.
- Update the CONFIG macro as follows:
  - Delete the CIPHR, TPFDF, and VEQR parameters.
  - If you want to add Apache code to the z/TPF system, add the APACHE parameter.
- Update the CORREQ macro as follows:
1. Delete the APSIZ24, APSIZ32, PTV, and SSPS parameters.
2. Code the CORREQ macro with the main storage parameters specified.
3. Code the CORREQ macro again with the memory configuration parameters specified. Update existing parameters to accommodate new working storage needs.

Notes:
1. You must code the CORREQ macro at least two times: the first time to define the main storage parameters, and the second time to define the memory configuration parameters. You can code as many as eight memory configurations.
2. The COMMON, FRAME, and IOBLK parameter names were changed to CMB, FRM, and IOB and are part of the memory configuration parameters.
3. You can code the CORREQ macro only when generating a base-only system, or when generating the basic subsystem in a multiple database function (MDBF) environment. Therefore, if you have the CORREQ macro coded in the SIP input deck of a subsystem, you must delete that entry.
   - Delete all GENFIL macro calls that specify 3350, 3375, or 9345 for the DEV parameter; these devices are no longer supported.
   - Update the GENFIL macro as follows:
     - Add the PDSNAME and TPFROOT parameters.
     - Delete the ANTPDS, ASMALL, ASMSPACE, ASMTYPE, BRELN, CCOMPTP, CFLAG, CLIBPFX, CLNGPFX, CNODE, CRUNTIM, CSRCH, EXCJOB, EXPRS, JCL, LINKALL, LISTAPE, OUTCL, PL1LK, PL1SR, REGN, USMAC, USOBJ, USPROC, USRCE, and XREF parameters.
   - Delete the BSCDRPS, BSCLN, BSCOP, PSLNS, and SLCAI parameters from the LINES macro.
   - Update the RAM macro as follows:
     - Add the APSIZE31 parameter
     - Delete the ECBNL and NFBACK parameters.
   - Update the ONLFIL macro as follows:
     - If you use the prime/duplicate module pairing (also referred to as even/odd module pairing) configuration, specify DUPMP=Y.
     - If you want to use keypoint 6 (CTK6) as a large keypoint to support greater than 3999 DASD, specify CTK6EXP=Y.

Notes:
1. If you use the standard IBM duplicate module pairing configuration and do not need to support greater than 3999 DASD, you do not need to update the ONLFIL macro.
2. Migration or coexistence between the standard IBM duplication module configuration and prime/duplicate module pairing is not supported. See \[Coexistence, migration, and fallback?on page 63\] for more information about coexistence and migration considerations.
   - Delete the DIAG, MAXDP, MAXEP, N2703, and SUBCH parameters from the NETWK macro.
   - Delete the XCHK parameter from the UTPROT macro.
   - Delete the BBSAT, BSNCT, INDSN, and SYNCLK macros.
Changes for the RAMFIL statements

Make the following changes to your RAMFIL statements. See z/TPF and z/TPFDF System Generation for more information about defining records, including guidelines for how many records to define based on your environment.

1. Define the following new records:

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CTKA</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKAB</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKALGF</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKA1 to #CTKA8</td>
<td>Keypoint staging area; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#CTKD</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKDB</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKDLGF</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTKD1 to #CTKD8</td>
<td>Keypoint staging area; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#CTK0</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK0B</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK0LG</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK01 to #CTK08</td>
<td>Keypoint record; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#CTK2</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK2B</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK2LG</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK21 to #CTK28</td>
<td>Keypoint staging area; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#CTK3</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK3B</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK3LG</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK31 to #CTK38</td>
<td>Keypoint staging area; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#CTK6</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK6B</td>
<td>Keypoint backup area; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK6LG</td>
<td>Keypoint record; define 12 ordinals.</td>
</tr>
<tr>
<td>#CTK61 to #CTK68</td>
<td>Keypoint record; define 12 ordinals for each record type.</td>
</tr>
<tr>
<td>#DBGDUMP</td>
<td>Debugger dump control and summary records; required only if you plan to capture and view dumps with the z/TPF debugger.</td>
</tr>
<tr>
<td>#FFSREC</td>
<td>Required only if you plan to use a fixed file system</td>
</tr>
<tr>
<td>#IF2A</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2AGCR</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2AI</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2AP</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
</tbody>
</table>
Table 3. New records (continued)

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#IF2API</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2APS</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2APSI</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2AS</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2ASI</td>
<td>Format-2 global record; required only if you plan to use format-2 globals.</td>
</tr>
<tr>
<td>#IF2G</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GCR</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GI</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GP</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GPI</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GPS</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GPSI</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GS</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IF2GSI</td>
<td>Format-2 global record; required only if you plan to use format-2globals.</td>
</tr>
<tr>
<td>#IPAT1–8</td>
<td>Program attribute table (PAT); required for all defined program bases.</td>
</tr>
<tr>
<td>#KLGFB</td>
<td>Keypoint backup area for loader general file.</td>
</tr>
<tr>
<td>#KPCTL</td>
<td>Keypoint pointer record for keypoint that are larger than 4 KB; define 12 ordinals.</td>
</tr>
<tr>
<td>#KPTCTL</td>
<td>Keypoint control record for keypoint that are larger than 4 KB; define 27 ordinals.</td>
</tr>
<tr>
<td>#PFSREC</td>
<td>Required only if you plan to use a pool file system</td>
</tr>
<tr>
<td>#RCAT5</td>
<td>Message router.</td>
</tr>
</tbody>
</table>

2. Change the RAMFIL statements to specify RECID=SPARE for the following record types that are no longer supported. Do not reuse these RAMFIL statements to define new records until you have completed your migration to the z/TPF system and you do not expect to fall back to the TPF 4.1 system or to run both a TPF 4.1 and z/TPF system in a loosely coupled environment.

Table 4. Records that are no longer supported.

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#APRGx</td>
<td>ADATA files; no replacement. Debug information now is stored in the file system instead of fixed file records.</td>
</tr>
<tr>
<td>#BRIDDEA</td>
<td>Recoup deactivated ID counts; replaced by #BRIDDE8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#BRIDSAV</td>
<td>Recoup ID counts record; replaced by #BRIDSA8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#BRIDTBL</td>
<td>Recoup ID counts record; replaced by #BRIDTB8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#BRIDTOT</td>
<td>Recoup ID counts record; replaced by #BRIDTO8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#CB8HD</td>
<td>MPIF hardware records; replaced by #HDREC for 32-way loosely coupled processor support (APAR PJ2778S on TPF 4.1 PUT 15).</td>
</tr>
</tbody>
</table>
Table 4. Records that are no longer supported. (continued)

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CCBRI</td>
<td>SNA conversation control blocks; replaced by #CCBRU for 32-way loosely coupled processor support (APAR PJ27785 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#DSCRI</td>
<td>GDS control records; replaced by #DSCRU for 32-way loosely coupled processor support (APAR PJ27785 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#GR34SR</td>
<td>Obsoleted by integrated online pool maintenance and recoup support (APAR PJ27469 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#KFBX0 to #KFBX254</td>
<td>Fallback extents records; no replacement.</td>
</tr>
<tr>
<td>#PSTCUR</td>
<td>Pool segment record; replaced by #PSTXCUR for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#PSTNEW</td>
<td>Pool segment record; replaced by #PSTXNEW for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#PVRx</td>
<td>Program version records; no replacement.</td>
</tr>
<tr>
<td>#RCATR</td>
<td>Routine control application table; replaced by #RCATU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#RV1RI</td>
<td>SNA resource vector table 1; replaced by #RV1RU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#RV2RI</td>
<td>SNA resource vector table 2; replaced by #RV2RU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#SATRI</td>
<td>SNA subarea address table; replaced by #SATRU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#SC1RI</td>
<td>SNA session control block 1; replaced by #SC1RU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#SC2RI</td>
<td>SNA session control block 2; replaced by #SC2RU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
<tr>
<td>#SONRPE</td>
<td>Pseudo directory records; replaced by #SONRPE0-7 for 32-way loosely coupled processor support (APAR PJ27785 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SONSP</td>
<td>Pools skipped address record; replaced by #SONSKP for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SRM31A</td>
<td>Recoup discrepancy counts record; replaced by #SRM31A8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SRM41A</td>
<td>Recoup counts record; replaced by #SRM41A8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SRM51A</td>
<td>Recoup counts history record; replaced by #SRM51A8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SRM61A</td>
<td>Recoup reference from record; replaced by #SRM61A8 for 8-byte file address support (APAR PJ28097 on TPF 4.1 PUT 15).</td>
</tr>
<tr>
<td>#SRTRI</td>
<td>SNA session recovery table; replaced by #SRTRU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
</tbody>
</table>
Table 4. Records that are no longer supported. (continued)

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#STPCR</td>
<td>Pool cycle up; replaced by #STPUR for 32-way loosely coupled pool support (APAR PJ27686 on TPF 4.1 PUT 14).</td>
</tr>
<tr>
<td>#TDATR</td>
<td>Tape device assignment table; replaced by #TDTDR for tape record migration (APAR PJ26577 on TPF 4.1 PUT 11).</td>
</tr>
<tr>
<td>#TLDMR</td>
<td>Tape label records; replaced by #TPLBL for tape record migration (APAR PJ26577 on TPF 4.1 PUT 11).</td>
</tr>
<tr>
<td>#WGTRI</td>
<td>Terminal control table; replaced by #WGTRU for infrastructure for 32-way loosely coupled processor support (APAR PJ27387 on TPF 4.1 PUT 13).</td>
</tr>
</tbody>
</table>

3. Increase the number of records defined for the following record types, as needed. The number of records that you need to define depends on your environment; see "z/TPF and z/TPFDF System Generation" for the guidelines about each record type.

Table 5. Increased records

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#FLOCK</td>
<td>z/TPF collection support file system (TFS) records; also used for storing z/TPF debugger information.</td>
</tr>
<tr>
<td>#INODE</td>
<td>z/TPF collection support file system (TFS) records; also used for storing z/TPF debugger information.</td>
</tr>
<tr>
<td>#IPL1</td>
<td>Contains IPLA and IPLB for image 1.</td>
</tr>
<tr>
<td>#IPL2 to #IPL4</td>
<td>Contains IPLA and IPLB for images 2 to 4; required only if you have multiple IPL areas defined (one record type for each additional IPL area).</td>
</tr>
<tr>
<td>#IRCMDF</td>
<td>z/TPFDF CRUISE record; increase from 3988 to 3989 ordinals per processor for more efficient hashing algorithm processing.</td>
</tr>
<tr>
<td>#IZERO</td>
<td>z/TPF collection support file system (TFS) records; also used for storing z/TPF debugger information.</td>
</tr>
<tr>
<td>#KBA</td>
<td>Keypoint backup area.</td>
</tr>
<tr>
<td>#KSA1 to #KSA8</td>
<td>Keypoint staging areas for images 1–8.</td>
</tr>
<tr>
<td>#OLD1 to #OLD8</td>
<td>E-type loader directory records; required to use the E-type loader.</td>
</tr>
<tr>
<td>#PROG1 to #PROG8</td>
<td>Program sequence records for E-type program area; required for all defined program bases.</td>
</tr>
<tr>
<td>#XPRG1 to #XPRG8</td>
<td>Program ordinal and text records for E-type programs; required for all defined program bases.</td>
</tr>
</tbody>
</table>

4. Decrease the number of records defined for the following record types as needed. This is not a required step; however, making these changes can save some storage.

Table 6. Decreased records

<table>
<thead>
<tr>
<th>Record</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#SRM41A8</td>
<td>Recoup records; decrease from 1947 ordinals to 531 ordinals (59 ordinals in each of 9 partitions).</td>
</tr>
<tr>
<td>#SRM51A8</td>
<td>Recoup records; decrease from 1947 ordinals to 531 ordinals (59 ordinals in each of 9 partitions).</td>
</tr>
</tbody>
</table>
Migrating to the z/TPFDF product

If you do not currently use the TPFDF 1.1.3 product, install the z/TPFDF product. See [z/TPF and z/TPFDF System Generation](#) for more information.

To migrate from the TPFDF 1.1.3 product to the z/TPFDF product, complete the following steps:

1. Forward fit your customized versions of the DBLCL and ACPDBE macros. Be aware of the following changes to the ACPDBE macro:
   - The following variables are no longer supported:
     - #ACPDBTS
     - #TPFDBID
     - #TPFDBFL
     - #TPFDDBS
     - #DBCOL1
   - The following variables were moved to the IDFEQ equate macro. Do not change the settings for these variables.
     - #ACPDB01 to #ACPDB10
     - #ACPDBFF
     - #TPFDB01 to #TPFDB10
     - #TPFDFF
     - #TPFDTS
     - #TPFDSCR
     - #DBCOL2
     - #DBCOLL
     - #NUM_SLOTS
     - #TABLE_SIZE
   See [z/TPF and z/TPFDF System Generation](#) for more information about the DBLCL and ACPDBE macros.

2. Forward fit any other user modifications. See "Customizing the code" on page 53 for more information.

3. Delete the following parameters from your DBDEF macro:
   - AUT
   - HIR
   - CI1
   - VFA
   - VI1
   These parameters were provided in the TPFDF 1.1.3 product for compatibility with previous versions; however, they are no longer recognized in the z/TPFDF product.

4. Convert your z/TPFDF C/C++ applications to be compatible with the GNU compiler collection (GCC). See [Migrating your application programs](#) for more information about application programming changes.

5. If you use a large number of W-type files and need to increase the maximum number of files that you can open, complete the following steps:
   a. Increase the value of #TPFDDBMO in the ACPDBE macro.
   b. Reassemble the UAO0, UBK0, UB40, and UB90 shared objects.
   c. Link the UTDF BAL shared object (BSO).

6. Review the migration considerations in the APEDITs for the following APARs that were released after TPFDF program update tape (PUT) 16 if you do not have them applied to your TPFDF 1.1.3 product already and make any
necessary changes to the z/TPFDF product:

*Table 7. TPFDF 1.1.3 APARs beyond PUT 16 with migration considerations*

<table>
<thead>
<tr>
<th>APAR Number</th>
<th>TPFDF PUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQ47099</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ50275</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ50524</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ50700</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ52738</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ53948</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ54398</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ56724</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ44993</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ55170</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ60572</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ61932</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ51955</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ74483</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ75887</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ79120</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ80141</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ80726</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ88266</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ90134</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ90430</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ91310</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ91496</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ92380</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ95747</td>
<td>PUT 17</td>
</tr>
<tr>
<td>PQ95818</td>
<td>PUT 17</td>
</tr>
</tbody>
</table>
Customizing the code

Complete the following steps to customize the z/TPF and z/TPFDF code to ensure that installation modifications remain fully functional after migration:

__1.  If you have modifications in the control program (CP), convert the modifications to run in 64-bit mode. Use the 64-bit assembler conversion aid (cvt64) to help you with these changes. See "Migration tools" on page 34 for more information.

__2.  If you use the user exit override address field or hard coded displacements when you build a user exit parameter list, you must change the code that builds the parameter list because the override address field has been expanded to 8 bytes. See the UXCMC macro in z/TPF General Services for more information about the format of the user exit parameter list.

__3.  See z/TPF System Installation Support Reference for information about user exits that are new, changed, or no longer supported and make any code changes that are appropriate.

__4.  Move any local modifications to the /local_mod directory. This allows you to isolate changes to the z/TPF system and z/TPFDF product code, and will make it easier for you to manage changes in the future.
Migrating your C library functions

Complete the following steps, as needed, to migrate TPF 4.1 library functions that are written in basic assembler language (BAL) to the z/TPF system. See “Application programming” on page 24 and “Migrating your application programs” on page 57 for more information about application programming considerations; also see z/TPF Application Programming for information about how to code your own library functions.

1. If you have TPF 4.1 library functions that must pass 64-bit pointers or pass more than five parameters, complete the following steps to migrate them for use on the z/TPF system:

   a. If the library function uses the TMSPC and TMSEC macros, change the code to use the PRLGC and EPLGC macros. While making that change, be aware of the following:
      • The TMSPC macro passes a pointer to the parameters in register 1 (R1); however, the PRLGC macro passes function parameter in R2 to R6. Ensure that the library function is coded to get the parameters from registers.
      • The TMSPC macro sets up the pointer to the current stack frame in R13; however, the PRLGC macro sets up the pointer in R8. Ensure that the library function does not refer to R8 or R13. Use the STACK parameter on the PRLGC macro to set up the stack frame pointer to a register other than R8.
      • The TMSPC macro uses the PPANAME parameter to define the function prototype; however, the PRLGC macro uses the ENAME parameter.
      • The PRLGC macro does not set up a base register; if you require a base register, you must code it explicitly.
      • The PRLGC macro does not use the LWS parameter. z/TPF functions must use the C stack frame to build the macro expansion that is needed to call z/TPF macro services.

   b. If the library function passes 64-bit pointers, change the code to use the 64-bit assembler instructions for processing pointers. See z/Architecture Principles of Operation for more information about 64-bit instructions.

   c. If the library function passes more than five parameters, change the code to address the C stack frame of the caller at label ICST_PAR to get the additional parameters. (The first five parameters are passed through R2 to R6.)

2. If you have TPF 4.1 library functions that are coded with either the TMSPC or PRLGC macro, consider the following and make any changes necessary:

   a. In the TPF 4.1 system, the C stack frame is mapped by the IDSDSA data macro. In the z/TPF system, the C stack frame is mapped by the ICSTK data macro.

   b. In the z/TPF system, add the CHECK parameter to the TMSPC or PRLGC macro for functions that call macro service routines that require macro authorization to ensure that the correct macro authorization is set.

   c. In the z/TPF system, the TMSPC and PRLGC macros generate trace hooks for functions that call macro service routines. If system
performance is a concern, you can code TRACE=NO on the TMSPC or PRLGC macro to prevent the macro from generating trace hooks.

d. If the TPF 4.1 library function is coded with the PRLGC macro and needs to run in 31-bit addressing mode in the z/TPF system, you must code AMODE=31 on the PRLGC macro. If you do not specify the AMODE parameter, the associated function will run in 64-bit addressing mode in the z/TPF system. If you code AMODE=31, be aware of the following:

- The function can pass no more than five parameters.
- The parameters must be valid 31-bit values.

3. If you have TPF 4.1 library functions that save or restore the address of the previous program in R8 without using the PBASC macro, delete that code. The z/TPF system code handles this function internally. However, if you use the PBASC macro in the library function, you do not have to change anything.
Migrating your application programs

Complete the following steps on your Linux system to migrate your application programs for use with the z/TPF system and z/TPFDF product. See “Application programming” on page 24 for general information about application programming considerations.

Note: The instructions and examples in this section assume that:
- The root directory named /tpf/z11 contains the IBM source code.
- The directory named /tpf/z11/base/util/migrationtools contains the migration tools that are included in these steps.
- A directory named /tpf/originalsource/applications contains your application program source code.
- A separate directory is created for the output of the updated source code to avoid overwriting the original files.

1. Update the header file names.

Conversion tools are available to help you update the header files. These tools are shell scripts that, in addition to renaming the files, make updates to the contents of the files. The input to each tool is the name of a directory. All files in the specified directory and subdirectories are processed. See “Migration tools” on page 34 for a list of the migration tools.

Run the tools that convert the $ characters before you run add_tpf_hdr.

Enter the following commands:

```
convert_hdr.sh /tpf/originalsource/applications
convert_src.sh /tpf/originalsource/applications
add_tpf_hdr.sh /tpf/convert_output/applications /tpf/z11/base/include/tpf
```

Notes:
- The output directory for convert_hdr.sh and convert_src.sh can be the same because the tools operate on different sets of files.
- These tools are written in the Korn shell and expect ksh to be installed in the /bin directory. If ksh has been installed in a different directory, modify the first line of all Korn shell scripts accordingly.
- Additional supporting files, named .README and .sed, also are provided for the Korn shell scripts. The .sed files must remain in the same path as these scripts.

2. Run the rmtrigraphs and rmbackslashes tools to make global changes for GNU compiler collection (GCC) considerations. Enter the following commands:

```
rmtrigraphs /tpf/add_tpf_output/applications
rmbackslashes /tpf/rmtrigraphs_output/applications
```

Note: These tools are written in Perl, so Perl must be installed in the /usr/bin directory. If Perl has been installed in a different directory, modify the first line of both tool files.
3. Complete the following steps to convert #pragma statements and _Packed directives:
   
a. Enter the following command:
   
   ```sh
tpf/rmbackslashes_output/applications
```
   This tool makes the following changes:
   
   • Surrounds #pragma map, #pragma filetag, #pragma checkout, and #pragma pack(reset) statements with guard macros. These macros process the statements based on whether the application is running on a z/TPF system or a TPF 4.1 system. For example, if the application runs on a z/TPF system, the #pragma map statements are changed to #define statements. If the application runs on a TPF 4.1 system, the #pragma map statements are left alone.
   
   • Converts #pragma pack (packed) statements to #pragma pack (1).
   
   • Adds guard macros (#ifdef__370__ and #endif) around _Packed directives. The tool creates an output file called _Packed.txt that contains a list of files that use the _Packed directive. Use this file to determine what files to update in step 3b. If there are no files that use the _Packed directive (that is, the _Packed.txt file is not created), skip step 3b and continue to step 4 on page 59.
   
   b. In each file listed in the _Packed.txt file, add __attribute__((packed)) around the outermost structure of the last _Packed structure. Surround the attribute with #ifdef__370__ and #endif to make the code compatible for single source. For example, make the following changes:
   
   ```c
#ifdef __370__
  _Packed
@endif
struct mystruct1 {
  ... some fields, structs, and unions
};
typedef
#ifdef __370__
  _Packed
@endif
struct mystruct2 {
  ...some fields, structs, and unions
} MYSTRUCT2;
```

   to:
   
   ```c
#ifdef __370__
  _Packed
@endif
struct mystruct {
  ... some fields, structs, and unions
}
#ifdef __370__
  __attribute__((packed))
@endif
;

typedef
#ifdef __370__
  _Packed
@endif
struct mystruct2 {
  ... some fields, structs, and unions
}
Note: The GCC compiler pads all structures at the end to make sure that they end on doubleword (8-byte) boundaries. This padding is included in the size of the structure. To correct this in addition to putting the `__attribute__((packed))` statement on the outermost structure, put the statement on all structures in unions inside the main structure.

4. Determine how the change in data type size (8 bytes to 4 bytes) affects your applications and make changes accordingly. See "Mapping data types" on page 81 for more information about the difference in data types.

If you want to follow the approach that IBM took with the system code, complete the following steps:

a. Enter the following command to convert all long data types to integer (int) data types:

```
convert_long_int.sh /tpf/rmbackslashes_output/applications /tpf/convert_long_output/applications
```

b. Change back selected fields to long data types as needed.

5. All pointers are automatically 8 bytes in size. Use the following rules to determine if a pointer should be redefined to be 4 bytes.

- For pointers in C structures that do not have equivalent assembler data macros (DSECTs) and are not written to file, leave these as pointers; that is, they are 8 bytes in length.
- For pointers in C structures that do not have equivalent assembler DSECTs but are written to file, investigate the use of these pointers. In general, pointers are not written to file, but changing the pointer to be 8 bytes may change the alignment of other data in the file, causing migration problems.
- For pointers in C structures that have equivalent assembler DSECTs, the storage for the pointer remains the same as in the DSECT. These pointers are 4 bytes in length unless the corresponding field in the DSECT is changed to be 8 bytes. To define the pointer as 4 bytes, update the C/C++ code to use 31-bit pointer type `__ptr32_t` or 31-bit pointer attribute PTR32ATT as follows:
  - Change all void pointers that must remain as 4 bytes to use the `__ptr32_t` data type. For example, change:
    ```
    void *myptr;
    ```
    to:
    ```
    __ptr32_t myptr;
    ```
  - Change all other (non-void) pointers that must remain as 4 bytes to use the PTR32ATT attribute. For example, change:
    ```
    struct mystruct * mystructptr;
    ```
    to:
    ```
    struct mystruct * PTR32ATT mystructptr;
    ```
  - Use the PTR32ATT attribute for 4-byte pointer type definitions. For example,
    ```
    typedef struct mystruct * PTR32ATT myptrtype;
    ```
• If the DSECT has an 8-byte pointer, do not change the C header structure.
• Define all new pointers as 8 bytes.

Notes:

a. The __ptr32_t data type and PTR32ATT attribute are defined in the bits/types.h header file.

b. In the TPF 4.1 system, some pointers are not correctly defined as pointers in C code and are simply coded as unsigned integers. In the z/TPF system, you must change these definitions to be pointers and they may require the use of the 32-bit pointer type or attribute.

c. The 31-bit pointer attribute does not support 31-bit function pointers. Function pointers must be 8 bytes.

d. See “Mapping data types” on page 81 for more information about the difference in data types.

6. The sysertime.h header file was renamed to sys/time.h. Change any include statements accordingly.

7. The GCC issues warnings when functions and variables are defined but not used. To prevent these warnings, do one of the following:
   • Delete the unused functions and variables.
   • Define the unused functions and variables with the __attribute__((unused)) attribute.

8. Review any hardcoded calculations; you might need to change some logic to compensate for calculations based on 4-byte pointers.

9. Some z/TPF typedefs were moved to tpf/types.h so that they can be used easily by all applications. The tpf/types.h header file cannot be included directly; to get tpf/types.h, include the sys/types.h header file.

10. Change the first parameter for any offsetof function calls to a data type. The offsetof function is coded as offsetof(TYPE, MEMBER), where TYPE is a data type and MEMBER is an element in that data type. For the TYPE parameter, GCC accepts only the TYPE data type. The IBM compiler accepts both a TYPE data type and a variable of type TYPE. This is an IBM compiler enhancement and is nonstandard. This type of coding produces an error when compiling with GCC. If your code passes a variable instead of a data type, GCC generates the following error:
parse error before ')' token

11. The TPF_regs structure is set up to pass 31-bit values to assembler code running in 31-bit mode. If your assembler program is modified to receive 64-bit values, you must modify the C program to specify that 64-bit values are being used in the TPF_regs structure.

To use TPF_regs in 64-bit mode, pass a pointer to the TPF_regs structure as a parameter to the TPF_regs_Call164 function and code the TPF_regs_Call164 function instead of TPF_regs in the parameter list of the assembler program. For example:

- To call assembler program ABCD with TPF_regs in 31-bit mode, code the function call as follows:
  ABCD(&regs);

- To call assembler program ABCD with TPF_regs in 64-bit mode, code the function call as follows:
  ABCD(TPF_regs_Call164(&regs));
Note: The program linkage automatically resets TPF_regs to 31-bit mode to prevent a 31-bit version of TPF_regs from accidentally being put into 64-bit mode. This can happen if a TPF_regs structure in 31-bit mode is allocated using the same area in memory that was used previously by a TPF_regs structure in 64-bit mode. Therefore, you must code the TPF_regs_Call64 function every time you want to call an assembler program using TPF_regs in 64-bit mode.

See z/TPF Application Programming for more information about how to use the TPF_regs structure.

__ 12. Run the maketpf.bsc.convert tool to convert your build scripts to makefiles. Enter man maketpf.bsc.convert for information about how to use this tool.

__ 13. Run the 64-bit conversion aid (cvtto64) under one or both of the following conditions:
   • You have application programs that must run in 64-bit mode.
   • You have application programs that use obsolete structured programming macro (SPM) names. The following table shows the old SPM names and the replacement names:

<table>
<thead>
<tr>
<th>Old SPM name</th>
<th>Replacement SPM name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CASENTR</td>
<td>#CAST</td>
</tr>
<tr>
<td>#ECASE</td>
<td>#ECAS</td>
</tr>
<tr>
<td>#ELOOP</td>
<td>#ELOP</td>
</tr>
<tr>
<td>#ELSEIF</td>
<td>#ELIF</td>
</tr>
<tr>
<td>#ESRCH</td>
<td>#EDO</td>
</tr>
<tr>
<td>#ORELSE</td>
<td>#OREL</td>
</tr>
<tr>
<td>#SRCEX</td>
<td>#DOEX</td>
</tr>
<tr>
<td>#STRTSRC</td>
<td>#DO</td>
</tr>
</tbody>
</table>

See “Migration tools” on page 34 for a summary of what this tool changes. See the cvtto64.README file for more information about how to install and run this tool.

Note: Most application programs can continue to run in 31-bit mode and do not need to be converted with this tool. Convert only the programs that must reference storage areas above the 2-GB bar or that use the old SPM names.

__ 14. If your assembler program must run in 64-bit addressing mode and uses the L instruction to load a fullword of data to a register, change the L instruction to a 64-bit instruction (such as LLGF or LLGTR) to zero out the upper 32 bits of the register. See z/Architecture Principles of Operation for more information about 64-bit instructions.

__ 15. Ensure that the first line of any application assembler code file, including macros and copy code, is not a comment or a blank line followed by a comment. The first line of the file must be a valid assembler command such as BEGIN, MACRO, SPACE, or EJECT.

__ 16. If you have code that accesses keypoint 3 (CTK3), change that code to use the necessary keypoint macros (GETKC, RELKC, and UPDKC). See z/TPF General Services for more information about these macros.
17. If you have application programs that use transfer vectors, ensure that the transfer vectors are coded on the BEGIN macro. See [z/TPF General Services](#) for more information about the BEGIN macro.

18. If you have application programs that specify PARENT=NO on the PROGC macro, or specify the PAT_PBI or PAT_DBI value on the progc function, change the code appropriately. The PARENT parameter and the PAT_PBI or PAT_DBI values are no longer supported.

**Note:** If your application specifies PARENT=YES on the PROGC macro, you do not have to change anything. This specification will be ignored in the z/TPF system.

19. Use the following floating point functions as necessary to ensure that all floating point data calculations are handled in binary floating point (BFP) format:
   - `tpf__fp_htob`
   - `tpf__fp_btoh`
   - `tpf__fp_hton`
   - `tpf__fp_ntoh`
   - `tpf__fp_bton`
   - `tpf__fp_ntob`

The `tpf__fp_htob` and `tpf__fp_btoh` functions are particularly helpful if you are doing a one-pass database conversion from hexadecimal floating point (HFP) to BFP.

The additional functions are available to allow you to write and maintain single source applications that run under both the TPF 4.1 and z/TPF systems. For example, if you know that a number in the database is stored in HFP, use the `tpf__fp_hton` to convert it to the native storage technique. That is, if the function is called from a TPF 4.1 application, the native storage technique is HFP; if the function is called from a z/TPF application, the native storage technique is BFP. The number can then be managed by using the native math functions and libraries. When the calculations are completed, call the `tpf__fp_ntoh` function to convert it back to HFP and store the number back to the database.

See the [z/TPF C/C++ Language Support User’s Guide](#) for more information about these functions.
Coexistence, migration, and fallback

The following information provides tasks and considerations related to:

- "Running TPF 4.1 and z/TPF" (coexistence)
- "Migrating from a TPF 4.1 system to a z/TPF system" on page 65
- "Falling back to a TPF 4.1 system" on page 66.

Running TPF 4.1 and z/TPF

Remember the following considerations if you plan to run a TPF 4.1 system and z/TPF system in the same loosely coupled complex:

- To install a z/TPF system, you must free image 1 to enable a full load with the general file loader.

  **Note:** Image 1 must be defined to use program area 1 and IPL area 1. See [z/TPF Program Management](#) for more information about the general file loader.

- Ensure that the first processor that you IPL as a z/TPF processor is the lowest ordinal in the complex because the processor with the lowest ordinal number does the keypoint copy. This ensures that the TPF 4.1 and z/TPF keypoints will be copied. If the processor with the lowest ordinal is a TPF 4.1 processor, the z/TPF keypoints will not be copied.

- When the z/TPF system is installed with the general file loader, additional images can be defined and loaded by using the image loader (called the auxiliary loader in the TPF 4.1 system). When you load an image for the first time, you must specify PROGCLEAR=YES in the loader input file.

- Because CTKX is not compatible between the z/TPF system and the TPF 4.1 system, if you want to use a TPF 4.1 image for the z/TPF system (or vice versa), you must copy or load all components, including CTKX.

- TPF 4.1 and z/TPF images cannot share IPL areas, program areas, or maintain logical copies of CIMR components.

- The loader general files must be release specific; that is, you must format the loader general file for the z/TPF system with the z/TPF IPLA. You cannot load z/TPF system components to a loader general file that was formatted with a TPF 4.1 IPLA.

- Do not load all keypoints to the online keypoint area. If you are running both a TPF 4.1 system and a z/TPF system, load only the following keypoints:
  - CTKA
  - CTKB
  - CTKD
  - CTKE
  - CTK0
  - CTK2
  - CTK3
  - CTKV

  **Note:** You must load all keypoints to the loader general file. See [z/TPF Program Management](#) for information about how to load system components to the loader general file and how to load specific keypoints to the online keypoint area.
You can define new format-2 global records to the z/TPF system at any time during the TPF 4.1 to z/TPF system migration; however, the format-2 global records will not be visible by any processor in a loosely coupled complex that is still running the TPF 4.1 system.

When running in a mixed TPF 4.1 and z/TPF loosely coupled complex, format-2 global record synchronization will affect only the migrated processors.

Do not use DASD symbolic device addresses (SDAs) that are greater than X'7FFF' because these addresses are not supported for coexistence.

Do not use tape SDAs greater than X'FFF' because the ZPROT DSP and ZTVAR ALTER commands will not run correctly. In addition, the capture function in the z/TPF system saves any tape addresses that are greater than X'FFF' in the keypoint in hexadecimal format; however, the TPF 4.1 display programs do not recognize this new format.

Do not use the WAIT parameter on the SYNCC macro because this parameter is not supported for coexistence.

You cannot run capture modules across releases. Consider the following example:

You have three processors (CPU-A, CPU-B, CPU-C) in a loosely coupled complex. CPU-A and CPU-B are running the TPF 4.1 system, and CPU-C is running the z/TPF system. If you start a capture on CPU-A, only CPU-A and CPU-B can capture modules. If you start a capture on CPU-C, only CPU-C can capture modules.

Run a keypoint capture from the z/TPF system to ensure that all keypoints are captured. A keypoint capture on the TPF 4.1 system will not capture z/TPF keypoints that are greater than 4 KB; however, a keypoint capture on the z/TPF system will capture all keypoints, including all TPF 4.1 and z/TPF keypoints.

IPL the last keypointed module in the z/TPF system to ensure that all TPF 4.1 and z/TPF keypoints, including those that are greater than 4 KB, are copied to the prime module. If you IPL the last keypointed module in the TPF 4.1 system, z/TPF keypoints that are greater than 4 KB will not be copied to the prime module.

You must run recoup from the z/TPF system under any of the following conditions:

– The TPFDF 1.1.3 product is not installed on your TPF 4.1 system. If this condition exists and you run recoup from the TPF 4.1 system, the new database that exists on the z/TPF system is not be recouped, which results in database corruption.

– The pool file system (PFS) is initialized in the z/TPF system.

– You are using other databases that exist on the z/TPF system but do not exist on the TPF 4.1 system. For example, the TPFDF database that is provided with continuous data collection (CDC). (CDC support was enhanced on the TPF 4.1 system with PUT 19 APAR PJ29925.)

Positive feedback databases are not supported across releases.

Consider using the IBM TPF Toolkit for WebSphere Studio for both the TPF 4.1 system and z/TPF system. While the debugger on the TPF 4.1 system can be used with either the IBM TPF Toolkit for WebSphere Studio or the VisualAge TPF for Microsoft Windows NT, the z/TPF debugger supports only the IBM TPF Toolkit for WebSphere Studio.

Use the floating point functions as necessary to ensure that all floating point data calculations are handled in binary floating point (BFP) format. See “Migrating your application programs” on page 57 for more information about changes.
related to floating point data; also see the \textit{z/TPF C/C++ Language Support User’s Guide} for details about the floating point functions.

- If you use continuous data collection (CDC), you must run a client for the TPF 4.1 system and another client for the \textit{z/TPF} system. You cannot monitor both systems with one CDC client. See \textit{z/TPF System Performance and Measurement Reference} for more information about how to run CDC.

- If you want to use your existing TPF 4.1 file system, enter the ZFILE command with no parameters specified when the system is in NORM state to ensure that the basic directory structure and files are there for the \textit{z/TPF} system.

### Migrating from a TPF 4.1 system to a \textit{z/TPF} system

Remember the following considerations if you plan to migrate your TPF 4.1 system to a \textit{z/TPF} system:

- If you want to use positive feedback when migrating to the \textit{z/TPF} system, enter \texttt{ZAPFS START}.

- After you have migrated the TPF 4.1 system to the \textit{z/TPF} system, you also can migrate format-1 global records to format-2 global records. Complete the following steps:
  __ 1. Update format-2 global user exit UGLM and load it to the online \textit{z/TPF} system.
  __ 2. Enter the ZGLBL GLOBAL DEFINE command with the \texttt{CONTROL-FORMAT1} parameter specified to define the new format-2 global record to the \textit{z/TPF} system.
  __ 3. Enter the ZGLBL GLOBAL INITIALIZE command with the \texttt{SOURCE-ASDEFINED} parameter specified to initialize the record.
  __ 4. Modify application programs to begin using the GLOBLC macro or the \texttt{tpf_g1}-type functions to access, keypoint, and synchronize the new global record. You do not have to migrate all applications at once because the data will continue to remain under the control of format-1 globals support.
  __ 5. After all applications that use the migrated global record have been modified, enter the ZGLBL GLOBAL MIGRATE command to migrate the format-1 global record to a format-2 global record that is fully system controlled.

  \textbf{Note:} Do not do this step until all processors in the loosely coupled complex migrated to the \textit{z/TPF} system.

  __ 6. After the global data is migrated to format-2 globals support, you can remove the format-1 global data from the \textit{z/TPF} system by completing the following steps:
  __ a. Remove the directory entry from the GL0Bx data macro.
  __ b. Modify the global storage allocator (GOA) records to remove the global record.
  __ c. Run the system test compiler (STC) to create a pilot tape with the new GOA layout.
  __ d. Enter the ZSLDR command to load the new pilot tape to the \textit{z/TPF} system.

- If you want to use your existing TPF 4.1 file system, enter the ZFILE command with no parameters specified when the system is in NORM state to ensure that the basic directory structure and files are there for the \textit{z/TPF} system.
Falling back to a TPF 4.1 system

If you need to fall back to a TPF 4.1 system, complete the following steps:

1. If you previously started positive feedback, enter ZAPFS STOP.
2. Enter the ZCYCL command to cycle the z/TPF system to 1052 state.
3. Enter the ZTOFF command to dismount all tape drives on the z/TPF system.
4. Enter the ZPSMS command to deactivate the z/TPF processor.
5. IPL the prime module and select the TPF 4.1 image.

When falling back to the TPF 4.1 system, be aware of the following conditions:

- If a processor has format-2 global records defined and falls back to a TPF 4.1 system, those format-2 global records will not be visible until the processor is migrated back to the z/TPF system.
- Recovery log processing is not supported across releases. If an error occurs when you enter the ZPSMS command to deactivate the z/TPF processor, enter the ZCRO command with the INIT parameter specified to initialize the recovery log on the TPF 4.1 system.
- If you modify the tape labels, any block sizes that were defined on the z/TPF system will be lost.
Part 3. Reference
Operating environment requirements and planning information

To ensure that your z/TPF system and z/TPFDF product perform correctly, you must establish the required operating environment. The following information describes the minimum system configuration requirements that are necessary, including:

- "Hardware"
- "Software (programming requirements)" on page 71.

Note: Some of the hardware or software products that are listed may no longer be supported by IBM.

Corequisite products
The z/TPF system and z/TPFDF product are corequisite products; that is, you must install both products.

Hardware
The following information describes the minimum hardware requirements for the z/TPF system and z/TPFDF product.

Servers
The z/TPF system supports the following servers:

- IBM @server zSeries 800
- IBM @server zSeries 890
- IBM @server zSeries 900
- IBM @server zSeries 990.

Loosely coupled considerations for these servers
Loosely coupled complexes require the following:

- Channel Redrive, which is activated when you set ESA/390 TPF mode in the activation profile for each image.
- Sysplex Timer attachment. For high-availability Sysplex Timers, the dual port Sysplex Timer attachment card is required.
- DASD control units require multi-path lock facility (MPLF). If you have a coupling facility (CF), DASD control units do not require MPLF.

Storage devices
The following provides information about storage devices such as tape units and direct access storage devices (DASD) and DASD control units.

Tape units
At least four tape units are required. The z/TPF system supports the following tape units:

- IBM 3490 Magnetic Tape Subsystem
- IBM 3490 Magnetic Tape Subsystem Enhanced Capability Models E01 and E11 (also known as the 3490E subsystem)
- IBM TotalStorage Enterprise Tape System 3590, including Models B, E, and H
• IBM TotalStorage Enterprise Tape Drive 3592.

**Tape control units**
At least one tape control unit is required. The z/TPF system supports the following tape control units:
• IBM 3490 Tape Control Unit
• IBM TotalStorage Enterprise Tape Controller 3590 Model A60
• IBM TotalStorage Enterprise Tape Controller 3592 Model J70.

The previous tape control units provide the following product features:
• IBM Enterprise Systems Connection (ESCON) channel adapter
• Fiber connection (FICON) channels.

**Tape libraries**
The z/TPF system supports the following tape libraries:
• IBM TotalStorage Enterprise Tape Library 3494
• IBM Magstar Virtual Tape Server (VTS), which includes support for ESCON and FICON connectivity.

**Direct access storage devices (DASDs) and DASD control units**
The z/TPF system supports the following DASD and DASD control units:
• IBM Enterprise Storage Server (ESS) Model 2105 (3990 Model 3 TPF mode)
• IBM TotalStorage DS6000
• IBM TotalStorage DS8000.

**Notes:**
1. The z/TPF system requires at least two DASD and one DASD control unit.
2. DASD control units require the multi-path lock facility (MPLF) in a loosely coupled complex. If you have a coupling facility (CF), DASD control units do not require MPLF.

**Interconnection devices**
The z/TPF system supports the following interconnection devices:
• IBM 3737 Remote Channel-to-Channel Unit Model 1, which allows IBM host systems to communicate with each other through public or private T1 facilities.
• IBM 9032 Enterprise Systems Connection (ESCON) Director, which provides for connectivity and switching among IBM ESCON channels and control units. Connect at least one IBM 9032 ESCON Director port to an IBM z/OS or IBM z/VM system for device support.
• IBM 9033 ESCON Director, which provides connectivity and switching among ESCON channels and control units. Connect at least one IBM 9033 ESCON Director port to an IBM z/OS or IBM z/VM system for device support.
• One IBM 9034 ESCON Converter Model 1 attached to one input/output (I/O) device with parallel channel adapters to ESCON channels. You can have more than one IBM 9034 ESCON Converter Model 1 to I/O device attachment in your complex.

**Communication controllers**
The z/TPF system supports the following communication controllers:
• IBM 3745, which requires z/OS Communications Server.
Consoles

The z/TPF system requires one system console and supports the following consoles:

- IBM 3215 Console support provided by platforms such as TPF Operations Server, including the IBM System/390 Channel to LAN Connectivity Card (9663-001)
- Local, non-SNA, DFT 3270 devices provided by control units such as the IBM 2074 or the OSA-Express Integrated Console Controller
- IBM Hardware Management Console (HMC) Operating System Messages Console
- IBM Hardware Management Console (HMC).

Loosely coupled complex

In a loosely coupled complex:

- Clock synchronization is required and can be achieved with the Sysplex Timer (STR)
- The z/TPF system must operate in ESA/390 TPF mode.

High Performance Option (HPO) feature

To run in a loosely coupled complex under the High Performance Option (HPO) feature, multi-processor interconnect facility interprocessor communications (MPIF IPC) is required with one of the following devices:

- IBM Enterprise Systems Connection (ESCON) Channel-to-Channel (CTC) device
- Fiber connection (FICON) CTC device.

Software (programming requirements)

The following information describes the minimum software requirements for the z/TPF system and z/TPFDF product.

Operating system requirements

- IBM z/OS Version 1 Release 3 (5694-A01) or later release, including High Level Assembler (HLASM) Release 5 and z/OS UNIX System Services (z/OS UNIX), installed on an eServer zSeries processor. See "Servers" on page 69 for a list of the supported processors.
- IBM Enterprise PL/I for z/OS Version 3 Release 3 (5655-H31), or later release
- IBM DATABASE 2 (DB2) Server for OS/390 Version 5 (5655-DB2), or later release
- Linux for zSeries (64-bit with 32-bit compatibility mode)

There are multiple ways to get Linux for zSeries:

Commercial distributions:

Commercial distributions are available from the following Web sites:

- IBM Linux Distribution Partners Red Hat [http://www.redhat.com]
- SUSE LINUX [http://www.suse.com]
- Turbolinux [http://www.turbolinux.com]
Commercially available Linux distributions can include the Linux operating system enabled for specific hardware platforms, an assortment of device drivers, routines for installation, and value add programs like Web servers and shells.

Typically, these distributions are delivered over the Internet or packaged on a physical deliverable. IBM suggests that these distributions be your first choice for production environments because of their availability of service and support from the Linux distributor or IBM support line.

**Build your own:**

IBM provides patches on the developerWorks [http://www.ibm.com/developerworks/opensource/linux390/index.shtml](http://www.ibm.com/developerworks/opensource/linux390/index.shtml) that are meant to be applied to the standard versions of the kernel, gcc, gdb, and so on. You can get these packages from the official Web pages where they are hosted (for example, [http://www.kernel.org](http://www.kernel.org) and [http://www.gnu.org](http://www.gnu.org)).

**Statement of direction**

IBM intends to provide High Level Assembler (HLASM) capability on Linux for zSeries before or at the time of the general availability of z/Transaction Processing Facility Enterprise Edition Version 1 Release 1.

This statement represents the current intention of IBM. IBM development plans and dates are subject to change or withdrawal without notice. Any reliance on this statement of direction is at the sole risk of the relying party and does not create any liability or obligation for IBM.

- GCC 3.4-based compiler with all TPF-thread updates applied (for example, gcc version 3.4-tpf-04r2-5 GNUPro compiler, available from Red Hat). See your IBM service representative for information about how to get a compiler for use with your z/TPF system.
- glibc and libstdc++ run-time libraries modified for z/TPF, which are available for download from [http://tpf.ncsa.uiuc.edu](http://tpf.ncsa.uiuc.edu).
- Korn Shell: pdksh (public domain Korn shell), which is available for download from [http://www.kornshell.com](http://www.kornshell.com).
- IBM z/TPF Database Facility Enterprise Edition Version 1 Release 1 (5748-F15)

**Note:** The z/TPFDF product is a corequisite for the z/TPF system.

To build the z/TPF system and the z/TPFDF product, you must have the following:

- IBM z/OS, IBM HLASM, and Linux, as specified previously
- The as and ld tools, which are available for download from the TPF Build Tools Web page at [http://www.ibm.com/tpf/download/bldtools.htm](http://www.ibm.com/tpf/download/bldtools.htm). You must install these build tools on z/OS UNIX.
Library system requirements

Terminology note

The term *library system* in the following information refers to any of the following:
- System configuration manager (SCM)
- Source code manager (SCM)
- Source configuration manager (SCM)
- Version control system (VCS)

The z/TPF library system must be hosted on a platform that is capable of supporting a hierarchical file system (HFS), such as Linux, UNIX, Windows, or z/OS UNIX System Services. If you plan to use the IBM TPF Toolkit for WebSphere Studio, the library system must have a Windows command-line interface.

Keep in mind that while the library system can reside on any HFS-capable system, the z/TPF system and z/TPFDF product are built on Linux. Therefore, it is important to have a library client that can communicate between the host library and the target Linux system. You must have a shadow of the library contents on the Linux system, or have a build process that extracts a copy of all the system code from the library system to the Linux system you build.

There are several products available to choose from, including:
- ClearCase (use this in snapshot mode only for best performance).
- ClearCase LT, which is a lower-cost alternative to ClearCase that works like the full ClearCase in snapshot mode.
- Concurrent Version System (CVS), which is open source.
- Subversion, which is an open source rewrite and enhancement of CVS.

*Note:* The previous list is only a subset of the available products.

z/Virtual Machine requirements

z/VM is not required to run the z/TPF system; however, if you plan to run z/TPF guests, IBM z/VM Version 4 Release 4 (5739-A03) or later release is required.

TPF Operations Server requirements

If you plan to use TPF Operations Server with the z/TPF system, you must have IBM TPF Operations Server Version 1 Release 2.01 (5799-GKX) or later release.

*Note:* If you plan to define the basic subsystem (BSS) with a name other than BSS, you also must apply APAR IC41752.

SNA-based communication requirements

For Systems Network Architecture (SNA)-based communication with remote users (terminal and application programs), the z/TPF system requires channel-attached controllers. You also can use channel-to-channel (CTC) or Enterprise Systems Connection (ESCON) CTC for communication between the z/TPF system and adjacent hosts.
PU 5 communication requirements
Connecting the z/TPF system as a PU 5 node to the SNA network requires channel-attached IBM 3745 Communications Controllers. You also can channel-attach the z/TPF system to IBM 3746 Model 900 (3746–900) Communications Controllers, which are connected to an IBM 3745. The IBM z/OS Communications Server is required to load and manage the IBM 3745 Communication Controllers.

You can use CTC or ESCON CTC for communication between the z/TPF system and adjacent hosts.

APPN communication requirements
Connecting the z/TPF system as an Advanced Peer-to-Peer Networking (APPN) node to the SNA network requires channel-attached routers that support the channel data link control (CDLC) protocol, such as:

- IBM 3745 Communications Controllers running IBM ACF/NCP Version 6 Release 2 or later release. The IBM z/OS Communications Server is required to load and manage the IBM 3745 Communications Controllers.
- IBM 3746 Model 900 (3746-900) or Model 950 (3746-950) Communications Controllers.

HPR communications requirements
Connecting the z/TPF system as a high-performance routing (HPR) node to the SNA network requires channel-attached routers that support the CDLC protocol and the HPR automatic network routing (ANR) feature, such as:

- IBM 3745 Communications Controllers running IBM ACF/NCP Version 7 Release 1 or later release. The IBM z/OS Communications Server is required to load and manage the IBM 3745 Communications Controllers.
- IBM 3746 Model 900 (3746-900) or Model 950 (3746-950) Communications Controllers.

X.25 communication requirements
Connecting the z/TPF system to remote X.25 devices requires channel-attached IBM 3745 Communications Controllers with the IBM NPSI or IBM FTPI feature installed. The IBM z/OS Communications Server is required to load and manage the IBM 3745 Communications Controllers.

ALCI communication requirements
Connecting the z/TPF system to remote ALC devices requires channel-attached IBM 3745 Communications Controllers with the IBM ALCI feature installed. The IBM z/OS Communications Server is required to load and manage the IBM 3745 Communications Controllers.

TCP/IP-based communication requirements
The z/TPF system can connect to Transmission Control Protocol/Internet Protocol (TCP/IP) networks through TCP/IP native stack support.

With TCP/IP native stack support, the z/TPF system can connect to the following devices:

- IBM 3745 Communications Controller with IBM NCP Version 7 Release 2 or later release.
- IBM 3746 Model 900 (3746-900) or Model 950 (3746-950) Communications Controller with the IP feature installed.
- Open Systems Adapter (OSA)-Express card with an Ethernet adapter.
You can access z/TPF and z/TPFDF product information in Hypertext Markup Language (HTML) format and Portable Document Format (PDF) from the IBM TPF Product Information Center, which is available from the TPF Web site at [http://www.ibm.com/tpf/pubs/tpfpubs.htm](http://www.ibm.com/tpf/pubs/tpfpubs.htm) and from the IBM TPF Product Information Center CD-ROM. Although IBM encourages you to use an information source that best matches your working style and environment, the IBM TPF Product Information Center is the preferred information source. You will find the most recent version of the product information on the TPF Web site.

The IBM TPF Product Information Center runs on Microsoft Windows 2000 and Windows XP operating systems using Microsoft Internet Explorer Version 6.0 or later, Mozilla-based browsers version 1.7 or later, or Firefox 1.0. Support for cookies and JavaScript must be enabled in the browser to ensure that all functions in the Eclipse help system are available. Otherwise, some functions of the help system might not be available.
Single source and migration APARs

The following information provides a summary of the authorized program analysis reports (APARs) that are available for the TPF 4.1 system and TPFDF 1.1.3 product to help with your migration:

- "Single source APARs"
- "Migration APARs" on page 78.

Single source APARs

There is a corresponding APAR for each change that you must make to an application program for it run on the z/TPF system. If you apply the APAR to the TPF 4.1 system or TPFDF 1.1.3 product, you can use the same application source for both the TPF 4.1 system and the z/TPF system.

**Note:** If you want to use any of the new z/TPF functionality, you cannot use the same source for applications on both levels of the system.

<table>
<thead>
<tr>
<th>APAR Number</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| PJ29218     | Adds support for the following application macros:  
- The LBASEC macro is a general macro that loads the base of the program that is currently processing for the issuing entry control block (ECB).  
- The LREGSC macro is a general macro that loads the contents of the register save area for the ECB into the requested registers.  
- The LOCKC macro is a system macro that locks a resource and prevents access to it by other l-streams. This macro is the same as the $LOCKC macro.  
- The UNLKC macro is a system macro that unlocks a resource previously locked by the LOCKC macro. This macro is the same as the $UNLKC macro.  

The BEGIN macro was updated to include parameters that are used in the z/TPF system. |
| PJ29436     | Adds support for changes to the activate_on_receipt (AOR) socket API. |
| PJ29575     | Adds the following data types:  
- _ptr32_t to explicitly represent 32-bit void pointers  
- _chptr32_t to explicitly represent 32-bit char pointers  
- _uptr32_t to explicitly represent 32-bit unsigned int pointers.  
The PTR32ATT macro definition also was added to assist in the declaration of these data types. Use the PTR32ATT macro to declare all explicit 32-bit pointers for any other pointer type. |
<p>| PJ29576     | Provides decimal data type support. |
| PJ29593     | Adds wrappers for header file name changes and the tpf directory. |
| PJ29630     | Adds time_t32 and size_t32 definitions. |
| PJ29640     | Adds support for the PRLGC, EPLGC, CSTKC, and PBASC macros. |
| PJ29691     | Adds support for the PNAMC, DEFBC, ENTRC, and ENTRNC general macros. |
| PJ29692     | Adds support for the CPROC and CALLC general macros. |</p>
<table>
<thead>
<tr>
<th>APAR Number</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJ29849</td>
<td>Adds support for the <code>tpf__fp_htob</code> and <code>tpf__fp_btoh</code> functions for floating point migration. Also, see PJ29980.</td>
</tr>
<tr>
<td>PJ29937</td>
<td>Moves the <code>gettimeofday</code> function declaration from <code>sysgtime.h</code> to <code>sys/time.h</code>.</td>
</tr>
<tr>
<td>PJ29948</td>
<td>Allows you to use the TLDR program to identify relocatable address constants (ADCONs), which are handled differently on the z/TPF system than they are on the TPF 4.1 system. To ensure single source compatibility between the z/TPF system and TPF 4.1 system, avoid relocatable ADCONs and use the support from this APAR to identify ADCONs to be removed.</td>
</tr>
<tr>
<td>PJ29957</td>
<td>Adds support for the time zone (TZ) environment variable.</td>
</tr>
<tr>
<td>PJ29969</td>
<td>Adds support for the SREGSC general macro and changes to the PNAMC general macro.</td>
</tr>
<tr>
<td>PJ29974</td>
<td>Adds an error trapping method for math functions.</td>
</tr>
<tr>
<td>PJ29980</td>
<td>Adds support for the <code>tpf__fp_hton</code>, <code>tpf__fp_ntoh</code>, <code>tpf__fp_bton</code>, and <code>tpf__fp_ntob</code> functions for floating point migration. Also, see PJ29849.</td>
</tr>
</tbody>
</table>
| PJ29984     | Moves the following definitions and declarations from `sysgtime.h` to `sys/time.h`:  
  * `FD_SET`
  * `FD_CLR`
  * `FD_ISSET`
  * `FD_COPY`
  * `FD_ZERO`
  * `tpf_select_bsd`
  * `struct fd_set`
  * `long fd_mask`
  * `NBBY`
  * `NFDBITS`
  * `howmany` |
| PJ30089     | Add support for the ERROR parameter on the CSONC macro. |
| PJ30136     | Corrects an error in the LREGSC macro that was added with APAR PJ29218. |
| PJ30189     | Allows TPFAR and z/TPFAR and applications to use `sqlint32` as a data type for SQL host variables. Use this data type for 4-byte integers that are used as host variables instead of a long data type to ensure the same behavior on the TPF 4.1 system and z/TPF system across supported DB2 precompilers. |
| PQ91889     | Adds wrappers for header file name changes for TPFDF. |

### Migration APARs

The following describes migration APARs that you might need to apply based on your specific configuration.

<table>
<thead>
<tr>
<th>APAR Number</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| PJ29358     | Provides support for prime/duplicate module pairing (also known as even-odd module pairing).  
  **Note:** If your TPF 4.1 system uses prime/duplicate module pairing, you **must** apply this APAR. |
<table>
<thead>
<tr>
<th>APAR Number</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJ29906</td>
<td>Corrects an error with password encryption processing. <strong>Note:</strong> If you use the z/TPF Internet mail server, you <strong>must</strong> apply this APAR.</td>
</tr>
<tr>
<td>PJ29083</td>
<td>Increases the size of the i-node, which enables the file system on the TPF 4.1 system to coexist with the z/TPF collection support file system (TFS) on the z/TPF system.</td>
</tr>
<tr>
<td>PJ30220</td>
<td>Corrects a problem that occurs during a restore of exception recording or logging tapes that were created after <strong>ZMODE 6</strong> was entered.</td>
</tr>
</tbody>
</table>
## Mapping data types

The following information provides a comparison of data types between the IBM C/C++ compiler and the GNU compiler collection (GCC). Use this as a reference when determining how to handle the change in data type sizes in the z/TPF system.

See [“Application programming” on page 24](#) and [“Migrating your application programs” on page 57](#) for more information about changes that you need to make for your application programs.

<table>
<thead>
<tr>
<th>Data type</th>
<th>IBM compiler</th>
<th>GCC</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>* (pointer)</td>
<td>pointer (4 bytes)</td>
<td>pointer (8 bytes)</td>
<td>In GCC, if you need to use a 4-byte pointer to be compatible with existing assembler code, use the PTR32ATT attribute or an existing 4-byte pointer data type. <strong>Attention:</strong> You must ensure that a 4-byte pointer never addresses anything above the 2-GB bar. If this occurs, the address will be truncated.</td>
</tr>
<tr>
<td>unsigned long, long, long int, and unsigned long int</td>
<td>integer (4 bytes)</td>
<td>integer (8 bytes)</td>
<td></td>
</tr>
<tr>
<td>long double</td>
<td>extended precision (16 bytes)</td>
<td>double (8 bytes)</td>
<td>Avoid using long double because there is no compatible data type available with GCC.</td>
</tr>
<tr>
<td>size_t</td>
<td>unsigned int (4 bytes)</td>
<td>unsigned long (8 bytes)</td>
<td>For compatibility, a typedef called size_t32 is now defined as an unsigned int (4 bytes).</td>
</tr>
<tr>
<td>ssize_t</td>
<td>signed int (4 bytes)</td>
<td>signed long (8 bytes)</td>
<td>For compatibility, a typedef called ssize_t32 is now defined as a signed int (4 bytes).</td>
</tr>
<tr>
<td>time_t</td>
<td>unsigned int (4 bytes)</td>
<td>unsigned long (8 bytes)</td>
<td>For compatibility, a typedef called time_t32 is now defined as an unsigned int (4 bytes).</td>
</tr>
<tr>
<td>clock_t</td>
<td>double (8 bytes)</td>
<td>long (8 bytes)</td>
<td>The size of this type has not changed, but the format has changed from a float to an integer type.</td>
</tr>
<tr>
<td>float, double</td>
<td>float (4 bytes)</td>
<td>float (4 bytes)</td>
<td>The size of these types has not changed, but the format has. With the IBM compiler, these data types are in Hexadecimal Floating Point (HFP) format. With GCC, they are in Binary Floating Point (BFP) format. This change in format affects C code that interfaces with assembler code or records on file.</td>
</tr>
<tr>
<td>struct TPF_regs</td>
<td>struct TPF_regs (total of 32 bytes)</td>
<td>struct TPF_regs (total of 72 bytes)</td>
<td>The size of the structure has more than doubled. Some code defines a pointer to TPF_regs and then points to EBW000 or EBX000. Change this to define TPF_regs as a local variable instead of a pointer.</td>
</tr>
<tr>
<td>enumerated data types</td>
<td>enum (1, 2, or 4 bytes depending on the value of the enumeration)</td>
<td>enum (4 bytes)</td>
<td>For GCC, all enumerated data types are 4 bytes. The size of structures that contain enumerated types will change and the relative location of the enumerations will shift. If the structure is in memory only, it can remain as it is. If the structure is written to DASD and cannot shift, change the enumerated type to a corresponding integer type (unsigned char or unsigned short) depending on the size you need. Instead of modifying the source code, you can specify the -fshort-enum compiler option when building your application to force enumerated types to use the smallest amount of storage possible.</td>
</tr>
</tbody>
</table>
Common errors and warnings

The following information describes some common GNU compiler collection (GCC) errors and warnings that you might see when migrating your code:

- “Compiler errors”
- “Compiler warnings” on page 87
- “Run-time errors” on page 93.

Compiler errors

Errors indicate problems that make it impossible to compile your program. GCC reports errors with the source file name and line number where the problem is apparent.

Header files not found

Problem: I received an error indicating that one or more header files were not found.

Solution: Some header files were moved to the TPF directory and need to have tpf/ added to the #include statement for that header. For example:

```
#include <tpfapi.h>
```

must be changed to:

```
#include <tpf/tpfapi.h>
```

This error also occurs if you reference a header file that is no longer supported in the z/TPF system. For example, header sysgtime.h is no longer supported. If you want to use the gettimeofday function, you must include the sys/time.h header.

Parse error

Problem: I received error message parse error before '})' token.

Solution: The cause for this error might be an actual parse error because of a missing parenthesis. However, if the line that this occurs on uses the offsetof function, the error might be caused by that function.

The offsetof function is coded as offsetof(TYPE, MEMBER), where TYPE is a data type and MEMBER is an element in that data type. For the TYPE parameter, GCC only accepts the TYPE data type. The IBM compiler accepts both a TYPE data type and a variable of type TYPE. This is an IBM compiler enhancement and is nonstandard. If you code a variable of type TYPE, GCC generates a parse error. To correct this, pass the data type instead of a variable with the offsetof function.

Function coded as a #define

Problem: I am receiving strange compile errors and warnings around a function that is actually a #define.

Solution: The IBM compiler allows you to implement a function as a #define and have the function return a value. For example, the IBM compiler allows the following:


```c
#define myfunction(parm1, parm2) {   
   int rc;   
   rc = parm1 + parm2;   
   return(rc);   
}
```

In GCC, you cannot have a `#define` return a value. To correct this, change the `#define` to static inline. The previous example would be coded as follows:

```c
static inline int __attribute__((unused)) myfunction(int parm1, int parm2) {   
   int rc;   
   rc = parm1 + parm2;   
   return(rc);   
}
```

**Initializer error**

**Problem:** I received error message Invalid Initializer or aggregate has a partly bracketed initializer.

**Solution:** Check the number of braces that you have surrounding an array initializer. With the IBM compiler, if you have an array initializer that initializes an array of structures, you only have to have braces at the outermost level. For example, the following is an array of structures, where each structure has three fields in it:

```c
struct mystruct array[] = {   
   item1a, item1b, item1c,   
   item2a, item2b, item2c,   
   item3a, item3b, item3c,   
   ...   
}
```

With GCC, each item in the array must be surrounded with braces. The previous example must be coded as follows for GCC:

```c
struct mystruct array[] = {   
   {item1a, item1b, item1c},   
   {item2a, item2b, item2c},   
   {item3a, item3b, item3c},   
   ...   
}
```

Another common occurrence of this error is initializing an array of structures to zeros. For example:

```c
struct some_struct structarray[] = {0};
```

must be changed to the following:

```c
struct some_struct structarray[] = {{{0}}};
```

**Ambiguous coding practices**

**Problem:** In C code, it is possible to code something in such a way that different compilers interpret it differently. Some types of coding are not covered by any standard; therefore, it is up to the compiler to interpret what is actually intended. For example, the following portion of code can be interpreted in several ways:

```c
int count = 0;
myfunction(++count, count);
```

The IBM compiler interprets this as follows:

```c
myfunction(1, 1);
```
The GCC compiler interprets this as follows:

\[ \text{myfunction}(1, 0); \]

Both are correct because no standard defines this behavior and the compiler determines how to interpret this code.

**Solution:** To correct this, you must remove the ambiguity of the code. Code the following to achieve \( \text{myfunction}(1, 1); \):

```c
int count = 0;
count++;
myfunction(count, count);
```

Code the following to achieve \( \text{myfunction}(1, 0); \):

```c
int count = 0;
myfunction(count+1, count);
count++;
```

**Undefined data type**

**Problem:** I received error message ISO C++ forbids declaration of `<variable_name>` with no type.

**Solution:** Ensure that all of your function parameters have the correct data type defined.

If a data type is not defined for a function parameter, the IBM compiler assumes that the parameter is an integer (int). For example, the count parameter in the following example would be handled as an int:

```c
void resetGetCount(const timewal inputTime,const count=0)
```

The GCC does not make this assumption; you must code the correct data type. To correct the previous example, code:

```c
void resetGetCount(const timewal inputTime,const int count=0)
```

**long to int conversion problems**

**Problem:** I received error message cannot convert 'long int*' to 'int*' for argument 'x'.

**Solution:** This can happen when a variable defined as long is passed to a function that was changed to use the int data type.

For the z/TPF system, some function parameters were changed from `long *` to `int *`. However, the calls to those functions might still be passing a `long *`. This causes an error because the called function will use the high-order 4 bytes of long and handle it as a 4-byte integer.

For example, consider the following function prototype. The last two parameters were changed from `long *` to `int *` when converting the code to GCC.

```c
void* T02_getBLOBwithBuffer(__TO2_pid*, void*, void*, int*, int*);
```

As a result, the following code receives this compiler error because the SEQ and BUFLEN variables are defined as type long.

```c
long seq;
MQSWEPT_HDR SweepInfo;              // header of swept BLOB
TO2_PID NextPid;                    // Next swept message PID
```
while ((rc == MQ2RC_OK) && !isPIDzero(CurPid)) {
    long BufLen = sizeof(MQSWEPT_HDR);
    if (TO2_getBLOBwithBuffer(&CurPid, pTO2Env, &SweepInfo,
                              &BufLen, &seq) == NULL) {

    ...
}

To correct this, change the data types of the variables:

```c
int seq;
MQSWEPT_HDR SweepInfo;  // header of swept BLOB
TO2_PID NextPid;        // Next swept message PID
```

while ((rc == MQ2RC_OK) && !isPIDzero(CurPid)) {
    int BufLen = sizeof(MQSWEPT_HDR);
    if (TO2_getBLOBwithBuffer(&CurPid, pTO2Env, &SweepInfo,
                              &BufLen, &seq) == NULL) {

    ...
}

In the previous example, the data type of a locale variable was changed. In some
instances, the incorrect variable might actually be an input parameter. For these,
make one of the following changes:

- Change the data type of the input parameter. Ensure that you do not break any
calling functions.
- Assign the input parameter to a local variable of the correct data type.

**Relocation error**

**Problem:** I received error message relocation truncated to fit :.

**Solution:** This is actually a binutils error, but the error occurs when you make a call
to tpf-gcc and specify the -shared command or the -or command, or both, to tell
the compiler to make a call to the linker.

This error is caused when your global offset table (GOT) size has exceeded 512
entries. When the GOT has reached this size, you must use the -fPIC option
instead of the -fpic option when compiling your code. You must add the -fPIC
option to the makefile for your library in all the correct areas because all source
code in the shared object must have this option.

You can mix and match the -fpic and -fPIC when your GOT contains fewer than 512
entries is allowed. However, you cannot mix and match these options when the
GOT has 512 entries or more.

**stdio.h error**

**Problem:** I received the following warning:
```
#error "Never use <bits/stdio.h> directly; include <stdio.h> instead"
```

**Solution:** Change the #include statement as shown in the following example:
```
#include <stdio.h>
```

Ensure that your application never directly includes an internal version of a header
file.
Compiler warnings

Warnings indicate other unusual conditions in your code that might indicate a problem, although compilation can (and does) proceed. Warning messages also display the source file name and line number, but include the following text to distinguish them from error messages:

```
warning:
```

Casting a pointer to an integer

**Problem:** I received the following warning:

```
warning: cast from pointer to integer of different size
```

**Solution:** Ensure that all pointers are defined correctly. Some pointers are referenced in assembler code and must be defined as 4-byte pointers using the PTR32ATT attribute or one of the predefined 4-byte pointer types. See the following for more information:

- bits/types.h
- “Application programming” on page 24
- “Migrating your application programs” on page 57
- “Mapping data types” on page 81.

If the pointer does not need to be defined as a 4-byte pointer, change the cast from unsigned int to unsigned long, or from int to long.

Class definitions and constructor variable parameters

**Problem:** I received the following warning:

```
warning: In constructor 'IVAViewBuffer::IVAViewBuffer()':
warning: member initializes for 'int IVAViewBuffer::BufferSize'
warning: and 'void*IVAViewBuffer::Buffer'
warning: will be re-ordered to match declaration order
```

**Solution:** Ensure that the class variable member definitions and the constructor initializer variable parameters are in the same order. Change the initializer order to match the class member definition order. For example, in the following code, the Instanceld initialization variable is not in the same order as it is in the class definition:

```cpp
class cQDEF
{
  protected:
    MQCHAR QName [MQ_Q_NAME_LENGTH];
    long InstanceId;
    MQCHAR4 StrucId;
    unsigned long Deleted :1;
    unsign long Modified :1;
    unsigned long Debug :1;
    unsigned long Protected :1;
    unsigned long DefPsist :1;
    unsigned long reserved_flag1 :27;
    MQLONG Version;

  ...

  protected:
  cQDEF(const MQLONG Type) : Deleted(FALSE),
  Modified(TRUE),
  Debug(FALSE),
  Protected(FALSE),
  DefPsist(FALSE),
};
```
To correct this, move InstancId in the cQDEF declaration to match the class definition.

```cpp
class cQDEF
{
    protected:
        MQCHAR QName [MQ_Q_NAME_LENGTH];
        long InstanceId;
        MQCHAR4 StrucId;
        unsigned long Deleted :1;
        unsigned long Modified :1;
        unsigned long Debug :1;
        unsigned long Protected :1;
        unsigned long DefPsist :1;
        unsigned long reserved_flag1 :27;
        MQLONG Version;

...}
```

```cpp
protected:\ncQDEF(const MQLONG Type) : Deleted(FALSE), Modified(TRUE), Debug(FALSE), Protected(FALSE), DefPsist(FALSE), reserved_flag1(0),
    InstanceId(0),
    Version(MQOD_CURRENT_VERSION),
```

### Defining counters in a loop

**Problem:** I received the following warning:

- warning: In function 'decimalResult'
- warning: operator/(const decimalProxy&, int)'
- warning: name lookup of 'd1'
- warning: changed for new ISO 'for' scoping
- warning: using obsolete
- warning: binding at 'd1'

**Solution:** This warning indicates that the counter was defined in the for loop itself (for example, for(int i=0; i+... and so on). Remove the int from the loop and define the counter at the top of the function.

### Assigning pointer to integer without cast

**Problem:** I received the following warning:

- 904: warning: assignment makes integer from pointer without a cast

**Solution:** In some headers there were 4-byte pointer fields that were changed from void * to unsigned integer. These fields remain 4-bytes. Use the __ptr32_t data type or define the pointer with the PTR32ATT attribute. See “Migrating your application programs” on page 57 and “Mapping data types” on page 81 for more information about using the new data type or attribute.
Unused variables or functions

**Problem:** I received the following warning:
warning: unused variable '<variable_name>'
warning: unused function '<function_name>'

**Solution:** If variable `<variable_name>` or function `<function_name>` is not used, it can be removed. If it is only used sometimes, you can use __attribute__((unused)). This attribute suppresses these warnings. For example:

```c
int __attribute__((unused)) myfunction(int parm1, long parm2) { ... }
long __attribute__((unused)) myvariable;
```

If there are a large number of unused variables or functions, which can happen with ported code, you can add the `-Wno-unused` compiler option to your makefile. This option suppresses all unused warnings.

Return with no value

**Problem:** I received the following warning:
warning: 'return' with no value, in function returning non-void

**Solution:** Sometimes, a function is supposed to return a value, but the return statement does not have a value on it. Investigate the function to determine if a value should be returned or if the function should not have a return value.

Control reaches the end of a non-void function

**Problem:** I received the following warning:
warning: control reaches end of non-void function

**Solution:** This warning is similar to the warning described in "Return with no value." If control reaches the end of a function and no return is encountered, GCC assumes a return with no return value. However, for this, the function requires a return value. At the end of the function, add a return statement that returns a suitable return value, even if control never reaches there.

Format code does not match variable type

**Problem:** I received the following warning:
warning: unsigned int format, different type arg (arg3)

**Solution:** Using the printf and sprintf functions, the format code specified does not match the variable type. Change the format type or cast the variable.

**Note:** When formatting using "%.*s", the variable representing the * character must be an integer. If it is a long type, it must be cast to an int.

No function prototype

**Problem:** I received the following warning:
warning: no function prototype for cs/cds/csg/cdsg functions

**Solution:** The compare-and-swap (cs, cds, csg, cdsg) prototypes were moved from the stdlib.h header file to the tpf/cmpswp.h header file. Include tpf/cmpswp.h in your source to correctly define the compare-and-swap functions.
Backslash and newline

**Problem:** I received the following warning:
warning: backslash and newline separated by space

**Solution:** The backslash (\) character is used as the continuation character to continue #define statements and strings to the next line. GCC expects the backslash character to be the very last character on the line. This warning indicates that there is a space after the backslash. Delete the space and any other characters that come after the backslash.

You can use the rmbackslashes tool to remove all backslashes from your programs.

See "Migration tools" on page 34 and "Migrating your application programs" on page 57 for more information.

Enumeration value not handled

**Problem:** I received the following warning:
warning: enumeration value <value> not handled in switch

**Solution:** This warning occurs if you have a switch statement based on an enumeration type. For example, you might have an enumerated type that has nine values, but your switch only tests three of those values:

```c
switch (myenum) {
  case val1:
    ...do something...
    break;
  case val2:
    ...do something...
    break;
  case val3:
    ...do something...
    break;
}
```

The logic in the previous example might be correct and you might not want to test for the other values. To remove the warning, add an empty default statement to the switch to handle the other values.

```c
switch (myenum) {
  case val1:
    ...do something...
    break;
  case val2:
    ...do something...
    break;
  case val3:
    ...do something...
    break;
  default:
    /* do nothing */
    break;
}
```

Problem with 2-digit year

**Problem:** I received the following warning:
warning: '%c' yields only last 2 digits of year in come locales
warning: '%y' yields only last 2 digits of year

**Solution:** This occurs for time formatting that uses only 2 digits for the year. Add the -Wno-format-y2k option to your compiler options.
Use of cast expressions

Problem: I received the following warning:
warning: use of cast expressions as lvalues is deprecated

Solution: This occurs when you cast the lvalue (left side) instead of casting the rvalue (right value). For example:
Incorrect: (void *) x = y;
Correct: x = (char *) y;

Type-pun problems

Problem: Why do I get warnings when I use type punning?

Solution: It is common to write data in one format of a union and read it from another, which is called type punning. The GCC does allow you to type pun without errors; however, if you typecast to a union, you are not using the union directly, and the GCC issues a warning that indicates strict typecasting does not allow this. The following are examples of incorrect and correct coding when you want to use type punning:
Incorrect:
void function_call(char);
myunion a_union;
char*cp; a_union.i=1;
cp=&(a_union.c[3]);
function_call(*cp);

Correct:
typedef union{
    int i;
    char c[4];
} myunion;
void function_call(char);
myunion a_union;
a_union.i=1;
function_call(a_union.c[3]);

Incorrect:
void fn_call(myunion*);
int i;
i = 1;
fn_call((union a_union*)&i);

Correct:
void fn_call(myunion*);
int i;
myunion a_union;
i = 1;
a_union.i = i;
fn_call(&a_union);

Problem: I received the following warning:
warning: dereferencing type-punned pointer will break strict-aliasing rules

Solution: This warning can occur when you cast a pointer from one type to a separate type because it can break strict-aliasing rules; that is, assigning a pointer of one type that resides in a pointer of another type is really just assigning two pointers of different types to point to the same raw data. If these expected types are sufficiently different, the compiler issues this warning when the code is optimized.
beyond level 1 because in its optimizations, the compiler determines that it is handling the same raw data as two different data types.

Consider the following example:

typedef struct mystructname {
    struct mystructname * prev;
    struct mystructname * next;
    int data1;
    char label1;
} mytypename;

int func1 (void ** address);
mytypename * func3(void);

int func2(void) {
    mytypename * parameterforfunc1;
    parameterforfunc1 = func3();
    return func1((void *) parameterforfunc1); // works with no errors
    // return func1((void **) &parameterforfunc1); // raises type-punned warning
    // because we broke strict
    // aliasing when we referenced
    // raw data by using the '&'

Note: If this type of warning occurs in ported code that you cannot fix, you can add the -fno-strict-aliasing compiler option to the makefile associated with the source that issues the warning.

Attribute packed statement ignored

**Problem:** Why does the compiler ignore the __attribute__((packed)) statement?

**Solution:** Ensure that the position of an __attribute__((packed)) statement is correct. The position is particularly important when the structure declaration also declares a particular instance of the structure. For this, if the position is incorrect, the compiler will ignore the attribute and issue a warning. Consider the following examples:

Incorrect:

typedef struct larry {
    <structure contents>
} LARRYSTRUCT __attribute__((packed));

Correct:

typedef struct moe {
    <structure contents>
} __attribute__((packed)) MOESTRUCT;

Problem generating the correct code

**Problem:** I received the following message:

note: if this code is reached, the program will abort

**Solution:** This message is issued when the compiler detects a condition where it cannot generate the correct code. The compiler also generates code that will cause a 000003 system error during run time. The code will branch to the middle of an instruction if you do not correct the C code.
For example, the following code:

\[
\text{glb}_\text{end} = \text{glb}_\text{start} + \ast\ast\text{(unsigned int, int)}\text{global}(\text{tagname, sizeparm});
\]

Generates the following warnings:

\text{warning: function called through a non-compatible type}
\text{note: if this code is reached, the program will abort}

In the previous example, the compiler generates the following assembler code. The \text{j.+2} instruction jumps to the middle of the next instruction.

\begin{verbatim}
145:/tpf51/bld/base/rt/cglmod.c ****
glb_end = glb_start +
133 .loc 1 145 0
134 0136 A7F40001 j +.2
135 .L7:
146:/tpf51/bld/base/rt/cglmod.c ****
136 .loc 1 153 0
137 013a 5810B118 l %r1,280(%r11) # tagname, tagname
\end{verbatim}

To correct this particular problem, code the following:

\[
\text{glb}_\text{end} = \text{glb}_\text{start} + \ast\ast\text{(long)}\text{global}(\text{tagname, sizeparm, NULL});
\]

### Run-time errors

The following information describes some run-time errors that you might see when migrating your code to GCC.

### DLM-style entry point

**Problem:** I am calling my CSO by the 4-character program name (that is, I have a DLM-style entry point), but the z/TPF system cannot find the entry point when I run my program.

**Solution:** Ensure that the entry point is coded on the CSO\_ENTRY statement in the makefile; also, ensure that the case and spelling of the function in the code and the function specified by CSO\_ENTRY match. See \textit{z/TPF Program Management} for more information about DLM-style entry points.

### Offsets do not match

**Problem:** The offsets between fields in the GCC structures versus the IBM compiler structures versus the assembler structures do not match.

**Solution:** There are several possible causes for this problem. You can use elfagger, an offline Linux program provided with the z/TPF system, and assembler listings to compare offsets between C/C++ structures and assembler data macros (DSECTs). The source for elfagger is in the base/util/src directory and the executable is in the linux/bin directory. Enter \texttt{man elfagger} on your Linux system for more information.

- Ensure that all fields defined are the same size. See \textit{Mapping data types} on page 81 for a list of C data types that have changed size between the IBM C/C++ compiler and GCC.
- 8-byte pointers and longs are forced to 8-byte boundaries in GCC. This can cause GCC to pad the structure to force boundary alignment where this padding did not exist in the IBM compiler or assembler structure.
- Bit fields are handled differently by the IBM compiler and the GCC. For example, if you defined the following field in the IBM compiler, it used only 3 bytes of storage:

\[
\text{int myfield :24;}
\]
In GCC, this field takes up 4 bytes of storage (it is a 4-byte integer), but only uses 3 bytes of that storage. To address this difference, pack the GCC version of the structure that contains the bit field. See “Migrating your application programs” on page 57 for information about how to pack a structure.

- If there is a pointer to a pointer to a 4-byte field, ensure that you use both the __ptr32_t data type and the PTR32ATT attribute. For example, consider the following 4-byte field in the ctk2 structure in the c_cck2sn.h header file:

```
POINTER *ck2rvt1a; //void **ck2rvt1a
```

To keep the size of this field 4 bytes long, code the following:

```
__ptr32_t * PTR32ATT ck2rvt1a
```

If the PTR32ATT attribute is left out, the size of this field will be 8 bytes and will not match the assembler structure.

- In the IBM compiler, enumerated data types use the smallest amount of storage possible based on the value of the enumerations. This means that the IBM compiler used 1, 2, or 4 bytes for an enumerated type. The GCC always uses 4 bytes. As a result, the size of structures that contain enumerated types will change and the relative location of the enumerations will shift. See “Mapping data types” on page 81 for more information about how to handle enumerated data types.

**Note:** The GCC pads all structures at the end to make sure they end on doubleword (8 byte) boundaries. This padding is included in the size of the structure. To correct this, in addition to putting the __attribute__((packed)) statement on the outermost structure, put it on all structures in unions inside the main structure.

### CTL-4 system error

**Problem:** I received a CTL-4 system error from CCNUCL during a FACS call.

**Solution:** Typically, a FACS call is done by loading fields R0, R6, and R7 in a TPF_regs structure and then calling the FACS function. Because these fields were 4 bytes in the TPF 4.1 system, the addresses loaded to R6 and R7 were cast to int. For example:

```
regs.r6 = (int) "#IBMMP4 ";
regs.r7 = (int) &ecbptr() ->ce1fa2);
```

However, with the GCC, addresses are 8 bytes. If the address of the IBMMP4 string loaded in R6 is above the 2-GB bar, the cast to int will truncate the address. This causes an address that is not valid and a CTL-4 system error from CCNUCL. To correct this, change the casts to long, as follows:

```
regs.r6 = (long) "#IBMMP4 ";
regs.r7 = (long) &ecbptr() ->ce1fa2);
```