IBM System z Personal Development Tool
Volume 1 Introduction and Reference

System z Development Tool

Full z/OS usage

Linux base

Bill Ogden

ibm.com/redbooks

Redbooks

International Technical Support Organization

IBM System z Personal Development Tool: Volume 1 Introduction and Reference

June 2013

SG24-7721-05

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

Sixth Edition (June 2013)

This edition applies to the IBM 1090 system (known as zPDT) that is available at the time of publication, corresponding to Version 1 Release 4 fixpack 1.

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication introduces the IBM System z® Personal Development Tool (zPDT), which runs on an underlying Linux system based on an Intel processor. zPDT provides a System z system on a PC capable of running current System z operating systems, including emulation of selected System z I/O devices and control units. It is intended as a development, demonstration, and learning platform and is not designed as a production system.

This book, providing an introduction, is the first of three volumes. The second volume describes the installation of zPDT (including the underlying Linux, and a particular z/OS® distribution) and basic usage patterns. The third volume discusses more advanced topics that may not interest all zPDT users. The IBM order numbers for the three volumes are SG24-7721, SG24-7722, and SG24-7723. An additional volume (SG24-7859) describes the use of zPDT in a Parallel Sysplex configuration.

The systems discussed in these volumes are complex, with elements of Linux (for the underlying PC machine), z/Architecture® (for the core zPDT elements), System z I/O functions (for emulated I/O devices), and z/OS (providing the System z application interface), and possibly with other System z operating systems. We assume the reader is familiar with general concepts and terminology of System z hardware and software elements and with basic PC Linux characteristics.

The author

This series of IBM Redbooks publications was produced by the zPDT development team, with assistance from many other people.

Bill Ogden is a retired Senior Technical Staff Member at the International Technical Support Organization, Poughkeepsie. He enjoys working with new mainframe users and entry-level systems.

The following people have contributed substantially to this project:

Keith VanBenschoten, IBM Poughkeepsie, was very helpful in establishing installation and startup processes for the 1090 and in providing test systems.

Theodore Bohizic, IBM Poughkeepsie, helped us understand command, design, and internal details.

Now you can become a published author, too!

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction

The IBM System z Personal Development Tool provides one or more System z processors (with several emulated I/O device types), based on a personal computer Linux environment. As the name implies, it is intended for development and related purposes, such as education and demonstrations. It lacks the RAS[1] and flexibility of a larger System z machine and is not intended or licensed for production use.

The IBM machine type identified with the System z Personal Development Tool is 1090.[2] System z operating systems detect machine type 1090, which is used for various ordering purposes. The IBM System z Personal Development Tool is often referenced as *zPDT* or a 1090 system. We use *zPDT* and 1090 as synonyms throughout this documentation.

IBM has long encouraged the use of several very small S/390®[3] environments for use by the IBM development community[4], and these have proven extremely useful. The zPDT offering provides a number of functions that extend the usefulness of very small System z development machines; these include the following:

- Large System z memory
- Full 64-bit System z operation
- QDIO channel operation for OSA-Express2 functions
- The more recent instructions that have been added to System z processors
- SCSI-attached tape drives, plus conversion utilities
- zAAP, zIIP, and IFL processors
- Simple installation and operation
- Cryptographic adapter functions
- Channel-to-channel (CTC) operations
- Coupling Facility functions
- Provision for HMC-like consoles

[1] Reliability, Availability, Support
[2] The version used with the RDzUT and RD&Tproducts is machine type 1091. There is no significant technical difference between the 1090 and 1091. This series of documents discusses the 1090, but the details apply to both versions.
[3] The reference to S/390 is for the historical context of this paragraph.
[4] In this context we primarily refer to the IBM PartnerWorld® for Developers organization (previously known as Partners in Development).

Providing these functions does not produce an environment equal to a larger System z, of course. Some aspects of a larger system are unlikely to be met in any very small environment; these include the ability to verify and enhance the scalability of a program under development, run application programs that require many hundreds of MIPS, exploit cross-LPAR functions, or use unique HMC/SE-related interfaces. A larger System z is needed for these areas of development. Likewise, a zPDT system is not advised for very fine-level performance tuning that is sensitive to memory location, cache functions, and pipeline optimization; larger System z machines have different characteristics than zPDT at this level.

The basic zPDT function consists of the zPDT software (processor functions, device emulators, utilities) and a hardware key device ("token") that is accessed in a USB port.[5] The hardware key determines the zPDT model that is used. The hardware key must be present (in a USB port) when zPDT is being used, but may be removed at other times. The System z architecture levels involved are indicated in Table 1-1.

Table 1-1 System z architecture levels

Release date	zPDT release	zPDT build level	z Architecture
2009, 2010	V1R1 "GA1"	39.xx	z800, z900
2011	V1R2 "GA2"	41.xx	z10
2012	V1R3 "GA3"	43.xx	z192
2013 fixpack May 2013	V1R4 "GA4.1" GA4+fixpack"	45.xx	EC 12

Three standard1090 models are available: L01, L02, and L03.[6] The model number indicates the number of System z CPs that may be defined and used by the 1090.[7] In most cases, the underlying Linux PC (that is used to install and run the 1090 system) should have *at least* one more PC core than the number of zPDT CPs used in the largest zPDT instance. At the time of writing, the standard 1091 token is equivalent to an L03 model.

The base Linux machine used for the zPDT must have sufficient memory. No specific size is required, but 8 GB should be regarded as a practical minimum and most zPDT systems have considerably more memory than this. Disk space is needed for emulated 3390 (or 3380 or FBA) volumes and a typical zPDT base machine will have *at least* 100 GB of disk space.

1.1 1090 and 1091 differences

The zPDT package available to Independent Software Vendors (ISVs) and for IBM internal use requires a 1090 token. It will not function with a 1091 token. All zPDT functionality is enabled with a 1090 token except that tokens larger than an L03 (three licenses) may not be used.[8]

The zPDT package available through RD&T, RDzUT, and other channels, requires a 1091 token and will not function with a 1090 token. Certain zPDT functionality is not available in a base 1091 system, but may be enabled through additional license features. These selected functions include Coupling Facility functions, the use of z/VM, and the use of 1091 tokens with

[5] It is possible to use a license server located in a remote Linux system instead of a 1090 or 1091 token in a local USB port. This is described in detail in the third document in this series (SG24-7723).

[6] The L03 version is standard for many RDzUT and RD&T customers.

[7] For this purpose, zAAPs, zIIPs, and IFLs are considered to be CPs. A 1090-L03, for example, can provide a maximum of three CPs+zIIPs+zAAPs+IFLs. Coupling Facilities (available only under z/VM) are not included in this count.

[8] However, more than one L03 1090 token may be used to provide more than three zPDT licenses.

more than three licenses. In addition, the more generalized IBM Rational License manager may be used in conjunction with a 1091 token. In all zPDT technical aspects the resulting zPDT usage is the same as it would be if using a 1091 token.

The license terms and conditions for the two packages are quite different (Table 1-2). License terms and conditions are not addressed in this series of publications. You must talk with an IBM marketing representative for details.

Table 1-2 1090 and 1091 comparisons

1090 (ISV usage)	1091 (RD&T, RDzUT usage)
1090 token only; 1091 token not usable	1091 token only; 1090 token not usable
Maximum of 8 CPs (with multiple tokens)	Standard maximum of 3 CPs; up to 8 with additional license feature
Coupling Facility usage (under z/VM)	Coupling Facility usage (under z/VM) only with additional license feature
1, 2, or 3 licenses in token. No way to order larger tokens.	3 license token standard; larger tokens available with special orders
Installed rpm name is z1090	Installed rpm name is z1091
z1090 rpm and z1091 rpm cannot be installed on the same machine	
z1090ver command	**z1091ver** command
“standard” z/OS AD-CD system	slightly modified z/OS AD-CD system
Other than the **z1090ver** and **z1091ver** commands, all other zPDT commands and functions are identical and may contain “1090” as part of the name. Documentation referencing 1090 also applies to 1091 except as noted here.	
Generalized RDz license manager may not be used	Generalized RDz license manager may be used (details from RDz)
A remote zPDT license manager and identity manager may be used. This is not the same as the generalized RDz license manager.	
Functional modules are installed in /usr/z1090/bin with additional materials in /usr/z1090/man and /usr/z1090/uim. zPDT instance files are created in a subdirectory named z1090 in the Linux userid home directory. There is no /usr/z1091 or ~/z1091 usage.	
z/VSE available (with proper license)	z/VSE not available

The differences, at the time of writing, between a 1090 and a 1091 system are summarized in Table 1-2.

The two packages (for 1090 and for 1091) are distributed separately. Each of the packages contains the following:

- Two support modules for the USB key. These are the same modules for both 1090 and 1091 tokens.
- Two zPDT *builds*: one for a Red Hat base and one for a SUSE base. (Specific Linux release levels are discussed in “Base configurations” on page 17.) The correct *build* is automatically selected when zPDT is installed.

The different versions for SUSE and Red Hat are due to slightly different libraries on the two distribution bases. Both are 64-bit programs and must run in a complete 64-bit environment.

Other than the limitations mentioned here, the zPDT functionality is the same for 1090 and 1091 tokens. This series of documentation usually addresses 1090 tokens, but the details also apply to 1091 tokens.

1.2 What is new (Version 1 Release 4 plus fixpack 1)

The current release, at the time of writing, is Version 1 Release 4 plus a “fixpack” released in May 2013 and is sometimes referenced as the GA4.1 release. This release includes the following changes:

- The relevant instruction set for the System z EC 12 processor is included. This is a major change for the base zPDT element. This includes significant new EC 12 functions:
  - Transaction Execution Facility
  - Runtime Instrumentation Facility
  - Decimal Format Conversion
  - 2 GB Page Frames
  - The flash memory function of EC 12 systems is *not* provided by zPDT at this time.
- 1090 and 1091 tokens may no longer be used interchangeably. A 1090 token works only with the zPDT package intended for 1090 tokens, and a 1091 token works only with the zPDT package intended for 1091 tokens. The IBM Rational license manager may be used in place of a 1091 token.
- Tokens with more than 3 zPDT licenses may be used with 1091 systems that are enabled for such usage.
- Two general virtualized environments may be used with zPDT. These are discussed in the third document in this series (SG24-7723-5 or later).
- While z/OS 2.1 had not been released at the time of writing, this zPDT release is expected to be compatible with it.
- Additional command scripts (**`aws_bashrc`** and **`aws_sysctl`**) are available to simplify zPDT installation. Also, there is now a **`1091ver`** command to match the older **`1090ver`**.
- The integrated consoles (3270 and ASCII) that are available with an HMC may be emulated with zPDT.
- A new command, **`z1090term`**, provides an ASCII console that can be connected to the integrated ASCII console interface.
- 3390 (and 3990) emulation has been upgraded to the level required for z/OS 2.1 (expected to be released in 2H2013).
- The remote license server that allows the USB keys to be installed in a central location has been improved.
- This release of zPDT has been built on RHEL 6.0, 6.1, and openSUSE 11.3 libraries. It is not usable with RHEL 5.x bases and is probably not usable with openSUSE 10.x bases.
- The cryptographic adapter functions provided by zPDT are now at the Crypto Express 4 (EC 12) level (CEX4C).
- A new level of the Coupling Facility code is included: level CFCC Level 18.
- zPDT includes a migration utility that may be used to copy 3390 volumes from a remote z/OS or z/VM system. This has been updated to function with older DASD on the “real” System z.
- A number of minor fixes are included in the GA4 release and fixpack level.
- Various performance improvements are included.

1.3 What is new (Version 1 Release 3)

The current release, at the time of writing, is Version 1 Release 3 (commonly known as GA3) and is dated March 2012. This release includes the following changes:

- The relevant instruction set for the System z 196 processor is included. This is a major change for the base zPDT element.
- A remote license server allows the USB keys to be installed in a central location. Multiple standard USB keys may be used (each with a maximum of three CP licenses) or nonstandard keys containing more licenses. Associated with this function is a Unique Identity Manager (UIM) that provides the same consistent serial number for the System z CPs in a given Linux machine. Details are included in the third volume of this series (SG24-7723-04). Several new commands are provided to manage these functions.
- This release of zPDT has been built on RHEL 6.0, 6.1, and openSUSE 11.3 libraries. It is not usable with RHEL 5.x bases and is probably not usable with openSUSE 10.x bases.
  - Various LSB warnings (Linux Standard Base) no longer appear during installation.
- The device map (devmap) used to define an instance of zPDT operation may now include Linux commands, with a method to control the timing of these commands. This function may be used to automate zPDT startup among other uses. In addition, environmental variables, **`include`** statements, and **`message`** statements are permitted in the devmap.
- The cryptographic adapter functions provided by zPDT are now at the Crypto Express 3 level.
  - z/OS releases earlier than 1.12 might require the fixes for APAR OA29839 to be applied.
- Performance enhancements are included. These are most noticeable for processor-bound programs, including the startup of the z/OS WebSphere Application Server.
- The license server function associated with token processing has been expanded to add significant security options. This is described in detail in the third volume of this series (SG24-7723-04).
- A new level of the Coupling Facility code is included: level CFCC D93G R17 SL4.8. This CFCC level is considerably larger than the CFCC included in the previous zPDT release and a larger z/VM guest is needed to use it. (We now recommend a z/VM guest size of at least 512 MB for a CFCC guest.)
- The Linux `/etc/profile.local` and `/etc/profile` files no longer require modification.
- The handling of LAN interfaces has been expanded to handle the new LAN interface names being used in later Linux releases. This involves changes to the output from the **`find_io`** command and changes to parameters for the awsosa device manager. These changes may require alternations in prior devmaps to match new path assignments.
- The zPDT **`stop`** and **`start`** commands have been extended to include **`stop all`** and **`start all`**.
- New RAS functions improve access to the USB key in rare cases where problems have been reported. The methods for starting the token interfaces during Linux booting have been enhanced.
- The maximum number of CPs (or the total of CPs, zIIPs, zAAPs, and IFLs) for a zPDT instance is now specified as eight. This does not indicate that an 8-way SMP is practical for zPDT, but indicates the maximum size of underlying zPDT control functions.
- A stricter statement of underlying PC processors ("cores") now exists. There must be at least *one more* core than the number of zPDT CPs in the largest zPDT instance running.[9]

  - An exception exists for a single core, which may be used with reduced zPDT performance.
- The use of USB 3 ports (for the USB key) is now supported.
- The 32-bit version of zPDT, previously available only within IBM, is no longer available.
- The SecureUpdateUtility must be run from the `/usr/z1090/bin` directory and must be run as *root*.
- Emulated DASD (3380, 3390) may be shared between instances of zPDT on the same PC. The performance of the zPDT locking involved in this sharing has been enhanced. (Note that this does not affect the need for sharing z/OS systems to provide serialization for access to the DASD.)
- zPDT includes a migration utility that may be used to copy 3390 volumes from a remote z/OS or z/VM system. The z/OS version of this utility previously included an *automatic restart* function that attempted to restart at the point of failure if a migration transfer was interrupted. This automatic restart function has been removed. If a volume migration is disrupted, it must be started again.
- When zIIPs or zAAPs are defined in a device map, at least one "cp" definition must precede the zIIP and/or zAAP in the processors statement.
- The output of the **`token`** command has been expanded to provide both zPDT license information and CP serial number information.
- Linux environmental variables may be used in device map specifications.
- Several minor commands have been added to permit an installation to administer zPDT tokens and license server configurations without switching to the Linux root userid.
- RDzUT customers may use more than three zPDT CPs, assuming sufficient zPDT licenses are available.
- The specification of **`ulimit -c unlimited`** for the zPDT operational environment may be reconsidered. This might be relevant for very large zPDT instances with, for example, 32 GB and larger System z storage specified.

1.3.1 Version 1 Release 2

Version 1 Release 2 (June 2011), known as the "GA 2.2 release," included the following updates. They are listed here as background information:

- zPDT has been adapted to later C libraries. (The earlier libraries created problems for recent Linux releases, such as Fedora 14.)
- Installation instructions are included to narrow the usage of an OSA emulation module that runs SUID to root. (This helps resolve a security concern.)
- A new **`pdsUtil`** command is included for all editing of certain z/OS partitioned data sets (PDSs) while running only under the base Linux.
- Additional information is included about installation and usage options for emulated OSA functions.
- The **`alcckd`** command has been changed such that it does not create Linux *sparse* files.
- The **`token`** command has been changed to display a token identifier of 1090 or 1091. The 1091 identifier indicates a token used for RDzUT.

[9] Previous zPDT releases could be used with the number of cores *equal* to the number of CPs in the largest instance. Changes to Linux kernel operation have dictated this change for zPDT. It may still be possible to run with the number of cores equal to the number of CPs in the largest instance, but this may not always be successful. In particular, running a two-CP instance on a PC with two cores may produce major performance problems.

- The Message Security Assist ("crypto instructions") has been enhanced to match the current z10™ level, including 256-bit key operations. This enhancement includes MSA levels 3 and 4.
- The serial number handling for a zPDT instance has been changed slightly. The change affects what happens if more than one token is involved. This function involves a new Linux-level service, uimd, provided by zPDT. (Note: This function was completely redesigned for Version 1 Release 3.)
- The installation instructions now specify that Linux 32-bit support functions are required. (This is so, even if you are using a 64-bit Linux.)
- Notes have been added about the use of the Customized Offerings Driver (COD) system.
- The log file permissions (for zPDT logs) have been tightened.
- A new **`listVtoc`** command has been added.
- New directions are included for updating the `/etc/sysctl.conf` file during zPDT installation.
- The **`z1090instcheck`** command has been updated.

1.4 Terminology

Terminology can be confusing in the computer business and especially when dealing with systems such as zPDT. In this documentation series, we use the following terminology:

- *System z Personal Development Tool* (usually known as a *1090* or *1091 system* or *zPDT*) is the name for the offering that includes the System z CP functionality and the USB hardware key. This does not include any System z software, such as operating systems.
- The *base machine* or *underlying host,* or *underlying Linux* is the Intel-compatible PC running Linux.
- *Machine type 1090* is the IBM processor type assigned. It is also the specific identifier for the USB hardware key needed to use the 1090. z1090 is used as a directory level for various libraries used by the 1090. (The token used for RDzUT and RD&T operation is known as a 1091. Unless noted otherwise, any discussion of 1090 usage also applies to the 1091. When used with a 1091 token, a System z CP still identifies itself as machine type 1090.)
- z/OS is used to refer to any recent release of the z/OS operating system. Likewise for z/VM®, and so forth.
- A device map, or *devmap*, is used to specify operational characteristics of zPDT. It is a simple Linux flat file.
- *Processor* or *core* normally refers to the Intel or AMD processors (cores) in the base machine. A two-core machine has two processors in this terminology, although both are in one hardware "processor" module. There is always some confusion in this terminology.
- *CP* refers to a general System z processor and is the major functional element of zPDT. By default, zPDT provides System z CPs. You may optionally convert a CP to a zIIP, zAAP, or IFL processor.[10]
- *Open Systems Adapter* (*OSA)* refers to an adapter on older S/390 machines but is sometimes used as shorthand for *OSA-Express*, *OSA-Express2,* and so forth. The

[10] Using more general System z terminology, the 1090 provides up to three PUs. By default, the PUs are characterized as CPs, but may be characterized as zIIPs, zAAPs, or IFLs instead. Throughout this document we generally refer only to CPs and this reference should be understood to include zIIPs, zAAPs, and IFLs when these are used.

operation of the original OSA adapters was often referenced as *LCS* mode. The zPDT system provides OSA-Express2 emulation (which can provide LCS mode and QDIO mode operation).

- Many Linux commands are shown throughout this series of documents. If the command is preceded with # (a hash or pound symbol) the command is entered in *root* mode; if the command is preceded with $ (dollar sign), it is not entered in *root* mode.
- The USB hardware key needed to enable zPDT operation is also referenced as a 1090 or 1091 hardware key, a zPDT token, a hardware token, and so forth.

The primary operational characteristic of zPDT, in which the instruction set of one computer platform (System z) is implemented through another platform (Intel or AMD) has a long history in the computer business. This design has been described with many terms, including microcode, millicode, simulation, emulation, translation, interception, assisted instructions, machine interface (MI) architecture, machine level code, and so forth. We attempt to avoid all this terminology and simply refer to the zPDT product.

1.5 1090 hardware token

A zPDT system is not functional without a 1090 or 1091 hardware token or key that typically connects to a USB port in your system.[11] The USB hardware keys are shown in Figure 1-1. A 1090 key is at the top of the illustration and should always have a tag attached to it. A 1091 key is at the bottom of the illustration and usually has a blue color code. The serial number of the 1090 key is printed on the tag. The serial number of the 1091 key is engraved on the back of the key.

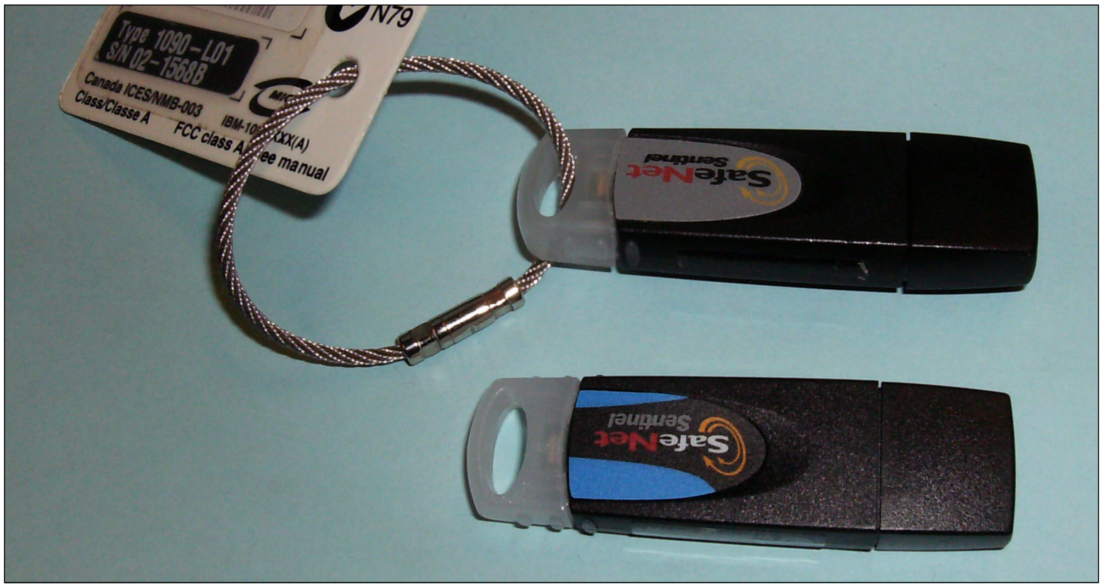

Figure 1-1 The 1090 and 1091 hardware keys

If the hardware key is removed while zPDT is operational, the operation will pause with a series of messages, ending with these:

```
AWSEMI318I zPDTA Heartbeat Missing for CPU 0
AWSEMI315I zPDTA License Unavailable for CPU 0
```

[11] Other arrangements involving license servers are also possible and are described in the third document in this series (SG24-7723).

Provided that the intervening time interval does not disrupt the operating system or application programs, zPDT operation may be resumed by connecting the hardware key again.

A USB hardware key is normally valid for one year after it has been initialized or *activated*. It may be re-initialized at any time, which normally extends the validity for a year beyond the date of the most recent re-initialization.[12] The procedure for initializing the key (or re-initializing it) depends on the channel you used to obtain your zPDT system. This may be through an IBM Business Partner or some other supplier.

More than one token may be used with a zPDT system.[13] For example, using two 1090-L03 tokens provides up to 6 CPs (or combinations of CPs, zIIPs, zAAPs, and IFLs). Coupling Facilities (available only under z/VM) are not counted for license purposes. The maximum number of CPs (including the specialized processors) for a 1090 zPDT instance is eight.

In general, we do not suggest zPDT usage with more than three or four CPs. The I/O capability of the underlying PC is a limitation, and various “SMP effects” substantially reduce the effectiveness of additional CPs above three or four. However, this performance determination is left to you.

1.5.1 Concurrent PC workloads

A System z processor, especially when running z/OS, must provide sufficient processing power to meet basic requirements. z/OS has various timers running to detect error situations. Sufficient processing power (for *each* CP, if multiple System z CPs are used) must be available to prevent these timers from expiring. Insufficient processing power can result in SPINLOOP, MIH[14] actions, OSA communication drops, or other apparent I/O device error problems.

A dedicated PC system (that is, not running any other significant workload) should not experience problems with typical developmental System z workloads. A “significant” workload is anything that consumes substantial processor cycles or ties up the disk drives over long time periods. This might be a Linux utility or an overcommitted virtual environment. A situation that creates unusually heavy PC disk I/O can create similar problems.

Reasonable care must be exercised even when extra base processor cores are available. For example, performing large Linux disk copies while the System z function is operational can effectively lock out normal System z work and create timeout situations.

It is possible to create “pathological” jobs that create I/O bottlenecks resulting in excessive MIH and other problems. We have not seen such situations in realistic development workloads, but the possibility exists.

1.6 zPDT functions

The zPDT functions include System z processor (CP) operation and the emulation of a variety of I/O devices. As a general statement, all the functions (instructions and I/O) needed to run current System z operating systems are provided.

[12] The extension period may differ depending on the IBM channel used to obtain the zPDT system.

[13] However, more than three CPs cannot be used with a 1091 system unless an additional license feature is obtained.

[14] Missing Interrupt Handler

System z character data is typically in EBCDIC, just as for any System z processor. Emulated disks and tapes typically contain EBCDIC data, although they logically contain whatever mix of EBCDIC, binary, ASCII, Unicode, or other formats that are produced by the System z operating system and applications. The key point is that there is no routine translation to the ASCII of the underlying host Linux system. The same binary data representation that is used on System z is also used on zPDT systems. This extends to fixed point, packed decimal, and all floating point formats. All zPDT data is in System z representation.

There are exceptions for emulated card readers and printers, where the character set involved is relevant and conversions between ASCII and EBCDIC are needed and are automatically provided.

Not all System z instructions and functions are available with zPDT. Instructions related to specific hardware facilities or optionally used by specialized programs might not be present. This excluded list includes:

BCPii functions
List-directed IPL
The accelerator function of cryptographic coprocessors
ETR
TOD steering
zBX functions
BCPii functions
CPU Measurement Facility
Asynchronous data movers
MIDAWs
Logical channel subsystems
Hipersockets
LPARs
Transport channel command functions
Flash storage
Multiple I/O paths

Parallel access to volumes (PAV) is not supported.

1.6.1 Emulated I/O

A zPDT system includes a number of *device managers*, each of which provides emulation for a related group of devices. A device manager can emulate multiple instances of its devices. The device managers are:

- awsckd - Provides emulation of 3380 and 3390 CKD disk devices. Each emulated device, such as a 3390-3, is contained in a single Linux file.
- awstape - Provides emulation of selected tape devices. Each tape volume is a single Linux file.
- aws3274 - Provides emulation of local, channel-attached 3270 terminals. Each terminal appears (to the System z operating system) as operating through a channel-attached non-SNA DFT IBM 3274 control unit.
- awsfba - Provides emulation for FBA disk devices (as used by z/VM and VSE).
- aws3215 - Provides emulation of a 3215 console.
- awsrdr - Provides emulation of a 2540 card reader, with functions to process both EBCDIC and ASCII data.
- awsprt - Provides emulation of a 1403-N1 or 3211 printer, including FCB emulation for 3211 functions. Automatic ASCII translation is provided.

- awsscsi - Emulates a mainframe tape drive using a SCSI tape drive. The only tested and supported drives are Fujitsu M2488E units (compatible with IBM 3490 and 3490E cartridges), IBM LTO3 and LTO units, and IBM 3592 (Fibre Channel interface) units.
- awsosa - Emulates an OSA-Express2 adapter, in either QDIO or non-QDIO mode. The hardware involved is an Ethernet adapter on the underlying PC.[15] This device manager can support TCP/IP operation. SNA operation is not supported at this time.[16] It can also support OSA/SF usage.
- awsoma - Is used to read CDs written in a special format known as OMA. In earlier days, VM was available in this format.
- awscmd - Provides a method to pass commands to the underlying Linux system and receive the command responses.
- awsctc - Provides emulated channel-to-channel operation via TCP/IP. The connection may be the same zPDT instance, another instance in the same PC, or an instance in a LAN-connected machine.

The emulated I/O support is summarized in Table 1-3.

Table 1-3 Emulated I/O summary

Manager	Control unit	Emulated device	Model
aws3274	3274	3270	
awsrdr	2821	2540 card reader	
awsprt	2821, 3811	1403, 3211 printers	
awsckd	3990	3380, 3390	1, 2, 3, 9[a]
awstape	3803, 3480, 3490	3420, 3422, 3480, 3490, 3490E, 3590	
awsfba	3990	9336	1, 2[b]
awsoma	3803	3422	OMA
awsscsi	3480, 3490	3490 (also 3480)	
awsosa	OSA	OSA	
aws3215	3215	3215	
awsctc	3088	3088	

a. Model 9 refers to 3390s. Actually, a 3390 with any valid number of cylinders may be defined and used, including EAV units.
b. The model emulated depends on the number of blocks defined, although z/VSE™ can force a model selection.

The design of zPDT allows for a large number of emulated I/O devices. The number is restricted, in practice, to better manage the memory and processing needed for emulated I/O. The current zPDT design allows a maximum of 1024 emulated I/O devices. This is often described as 1024 *subchannels*.

[15] Wireless can be considered an Ethernet adapter.
[16] It may be possible to initiate SNA operations (in non-QDIO mode) but this usage has not been tested and is not supported by IBM at this time.

1.7 Operational overview

This section provides a brief introduction to the zPDT definition and operational structure.

1.7.1 Linux userids

In principle, any Linux userid may be used to install[17] or operate zPDT, with the exception that the zPDT operational Linux userid cannot be longer than eight characters. All our examples assume userid *ibmsys1* is used. The zPDT system uses several default path names that are related to the current Linux userid. Control commands for zPDT, such as the `ipl` command, must be issued from the same Linux userid that started the zPDT instance.

In principle, a different userid could be used to create a completely different zPDT operational environment, with different control files, and so forth. Also, multiple Linux userids *must* be used when running multiple zPDT instances concurrently. We use *ibmsys2* and *ibmsys3* as examples of these additional userids.

Our Linux operating systems automatically create home directories for userids in the format /home/<userid>. For example, the home directory for userid ibmsys1 is /home/ibmsys1. It is possible to specify a different home directory for a userid. Throughout this document we use */home/<userid>* or */home/ibmsys1* to indicate the home directory for the userid.

1.7.2 zPDT instances

Logging into Linux and starting a zPDT operation creates an *instance* of zPDT usage. This instance might have one, two, or three System z CPs associated with it,[18] depending on the 1090/1091 model expressed in the 1090/1091 token and the parameters in the devmap. If you then log into Linux with a second Linux userid, and start another zPDT operation, this creates a second instance. Multiple instances means that multiple, independent zPDT environments are run in parallel. The total number of CPs in all concurrent instances cannot exceed the number allowed by the token model number.[19]

A 1090 model L03 can have up to three System z CPs (or mixtures of CPs, zIIPs, zAAPs, and IFLs). These could be used for three zPDT instances, each with a single CP and separate System z memory[20], and a separate System z operating system. Alternatively, a single zPDT instance could be used with one, two, or three CPs; this is the more likely usage for most zPDT users. The use of multiple CPs is subject to the following restrictions and considerations:

- The number of defined CPs (including zIIPs, zAAPs, or IFLs) in one zPDT instance must be one less than the number of processor cores on the base Linux system. For example, a W500 mobile computer with a dual core cannot have more than one CP defined in an instance.
- Full zPDT operation can use more processor cores in the base system than there are System z CPs defined in any one instance. The additional processors are used for I/O, to help prepare System z instructions for use, and for non-zPDT Linux processes.

It is important to understand that the zPDT license controls are on the number of System z CPs (or zIIPs, zAAPs, or IFLs) that are in concurrent use, and not on the number of base PC

[17] Part of the installation process must be done as *root*, but the initial login should be with the userid that will be used to operate zPDT.
[18] Or more, if multiple standard tokens are used or larger nonstandard tokens. We ignore these exceptions in the following discussion.
[19] Or by the total number of licenses from multiple tokens, with a design limit of eight for any given zPDT instance.
[20] The combined System z memory is subject to the later discussion about memory.

system processor cores that are being used. With a 1090-L03 token, there is an absolute maximum of three instances (each with a single CP) with each running, for example, a different release of z/OS.

Table 1-4 may help make this clearer. It indicates the number of zPDT instances (in concurrent operation) and the possible CP arrangement for each 1090 model type. A CP can be converted to a zIIP, zAAP or IFL, but this does not change the maximum. For example, a 1090-L03 could have a single instance with one CP plus one zIIP plus one zAAP; this would exhaust the number of CPs available with a 1090-L03 and additional concurrent instances would not be possible. The same limitations apply to 1091 tokens.

Table 1-4 Possible CP configurations

Model	One instance	Two instances	Three instances
1090-L01	1 CP	Not possible	Not possible
1090-L02	1 or 2 CPs	1 CP each	Not possible
1090-L03	1, 2, or 3 CPs	1 CP each, or 1 CP in one and 2 CPs in the other	1 CP each

It is possible to use more than one 1090 token. For example, a machine with two model L03 tokens would have a maximum of six CPs. However, IBM has not extensively tested the usage of multiple tokens. There is a clear “multiprocessor effect” present and the advantages of more than four or five CPs may be marginal, depending on the nature of the workload. Also, the I/O limitations of the underlying PC become very relevant when using more than three CPs.

In basic usage, emulated I/O devices are unique to a zPDT instance. However, there are advanced zPDT options that permit sharing emulated I/O devices among multiple instances. The minimum number of base processor cores, as stated earlier, is one more than the maximum number of CPs in any instance. Other than this, there is no association of particular base processor cores to CPs.

The remainder of this document, and all the discussions in Volume 2 of this series, focus on single instance operation. A chapter in Volume 3 of this series provides setup and usage instructions for multiple zPDT instances.

1.7.3 Small system example

The environment for a zPDT instance is defined by a device map, commonly known as a devmap. The following devmap describes a simple System z:

```
[system]
memory 5000m                    # emulated System z to have 5000 MB memory
3270port 3270                   # tn3270e connections will specify this port
processors 1

[manager]
name awsckd 0001                # define two 3390 units
device 0a80 3390 3990 /z/SARES1
device 0a81 3390 3990 /z/WORK02

[manager]
name awstape 0020
device 0580 3480 3480 /z/SAINIT  #tape drive with premounted tape volume
device 0581 3480 3480           #tape drive with no premounted volume

[manager]
name aws3274 0300              # define two local 3270s
device 0700 3279 3274
device 0701 3279 3274
```

Device managers (such as awsckd, awstape, and aws3274 in the example) are the zPDT programs that emulate various device types. The number after the device manager name is an arbitrary hexadecimal number (up to four digits) that must be different for each *name* statement.

Device statements in the devmap specify details such as a device number ("address"), device type, the Linux file used for volume emulation, and various other parameters. The volume mounted at an address can be specified or changed with the **awsmount** command while the 1090 is running. In this example, the emulated tape volume in Linux file /z/SAINIT is already mounted when zPDT is started. We could change the volume (while zPDT is running) with an **awsmount** command that specifies a different Linux file. (The files must be in the proper emulated format, of course.) This corresponds to changing a tape volume on a tape drive.

1.7.4 1090 console

A zPDT system is operated from Linux command lines. This operation could be done remotely through telnet or ssh connections. A graphics connection is not needed.

There is no dedicated console program for sending commands to an operational zPDT environment.[21] All zPDT commands are Linux executable files that are entered from a Linux shell prompt. The commands *require* that the zPDT instance be started by the same Linux userid that issues the subsequent zPDT commands for that instance. For example, if Linux userid ibmsys1 starts zPDT then only Linux userid ibmsys1 can issue an **ipl** command. The **ipl** command is a Linux executable file, supplied with the other executables that constitute the zPDT package.

1.7.5 Performance

Discussing performance for zPDT is difficult for several reasons, including the following:

- The performance depends on the power of the underlying hardware and this changes frequently. Performance is not only related to the clock speed of the underlying processor (such as 2.4 GHz for an Intel processor) but is related to the memory design and the pipelining, caching, and translation design of the underlying processor. For example, substantial performance changes may be observed by simply reordering program instructions in an optimum way for the underlying processor.
- Linux performance (including applications such as zPDT) can be greatly influenced by how the Linux disk cache (and swap file) is performing.
- The number of CPs used by zPDT has an obvious effect, as do the number of cores in the PC processor.
- Every new release or update of zPDT can change performance.
- The System z instruction mix and memory reference pattern has a profound impact on performance—probably a greater impact than is observed on a larger System z.
- MIPS (million instructions per second) is a rather discredited metric, although it is still informally used with small System z machines. Any MIPS number is *very* dependent on

[21] Do not confuse zPDT commands with z/OS operator commands.

the nature of the workload and the Linux configuration. MSU numbers are less variable, but again depend on the nature of the workload.

- I/O performance must be considered. For example, all emulated disk and tape operations for a 1090 on a mobile computer are from the single (relatively slow) mobile computer disk drive. Workloads with modest I/O loads (when run on a larger System z machine) might be completely I/O-bound on a mobile computer-based 1090 system.

As an example, zPDT on a Lenovo W520 mobile computer provides reasonable performance for IPLing and running z/OS, with typical TSO and batch usage, small DB2® usage, and so forth. Using emulated local 3270 connections, reasonable performance might be maintained for a number of such users. The general look-and-feel for such usage generally provides subsecond response typical of smaller System z installations.

The zPDT design goals are based on the assumption that it is the only significant application running on the host machine. The impact of additional applications (even trivial functions, such as a game) is most significant for Linux memory management. This can be considerably more important than the extra CPU cycles taken by another application.

z/VM may be used with zPDT. The performance of guest operating systems under z/VM (such as z/OS running under z/VM) is greatly influenced by the use of the SIE instruction. On a larger System z machine, this instruction provides a “microcode assist”[22] for many of the virtualization functions performed by z/VM. Most SIE functions are provided by the 1090, but there is no direct equivalent of a “microcode assist” level and the virtualization performance boost provided by SIE is modest. Informal measurements indicate that traditional z/OS workloads, with z/OS running under z/VM on the zPDT, perform at about .75 (or better) of their performance when run natively (without z/VM) on zPDT. As usual, the exact performance ratio depends on the nature of the workload.

[22] This is the common terminology for SIE operations, although the actual implementation may be much more complex than implied by this statement.

2

Base configurations

There is a range of personal computer systems and Linux distributions that might be used for zPDT. These configurations change over time, due to frequent personal computer hardware advances and new Linux releases. As a general statement, zPDT should work with any modern Intel-compatible processor that is fully supported by the recommended Linux distributions.

The combination of the base Linux, zPDT operation, and z/OS operation (for example), with associated LAN usage and emulated I/O devices, produces a very complex environment. IBM has tested zPDT functions extensively, but with a limited number of PC hardware configurations.

2.1 zPDT base configurations

The 1090 formal IBM license statement regarding base systems is as follows:

> "The Program may be used on the following systems which are running versions of Linux as specified in the Program's read-me file: IBM System x® 3500 M1, 3500 M2, 3500 M3, 3650 M1, 3650 MM2, or 3650 M3; Lenovo Thinkpad W Series; or systems otherwise approved by IBM.

The license agreements may contain reporting requirements that must be understood by the user. These are not covered in this document and may be reviewed with your IBM representative.

2.2 Hardware and software levels

Both PC hardware and base Linux software change frequently. zPDT changes are needed to maintain a reasonable level of compatibility. zPDT is not intended to be compatible with all levels of Linux or with all available PC hardware. An informal guideline for both hardware and software might be "not too old and not too new."

Base Linux

zPDT Version 1 Release 4 (December 2012) has been built on openSUSE 11.3, SLES 11 Service Pack 2, and RHEL 6.0 and 6.1. These are the "supported" base Linux releases.

Informally, the SUSE version has been used on all releases of openSUSE 11, and with openSUSE 12.1 and 12.2. This zPDT release cannot be used with openSUSE 10.x releases.

The RHEL version of this release cannot be used with RHEL 5.x releases. It must be used with RHEL 6.0, 6.1, 6.2, 6.3, or possibly later releases. It has been used informally with Fedora 17.

SLES releases and Fedora releases corresponding to the openSUSE and RHEL releases mentioned here have been informally used but not in any formal test.

Later Linux distributions may require you to make administrative adjustments. For example, at the time of writing some Linux distributions require additional commands to provide optimum OSA performance.

Important: A useful System z installation, even as small as a zPDT system, can represent a major investment for the owner. The zPDT development team assumes serious users will select one of the supported Linux bases (RHEL or SLES) that have been extensively tested with zPDT. Over time zPDT will follow general Linux developments and changes, but constantly chasing the latest Linux distributions is not a primary zPDT goal.

See Chapter 15 in the third document in this series (SG24-7723-05, or later) for information about zPDT usage in a virtual environment.

A suitable 3270 emulator is needed. Many current Linux distributions may not include the x3270 package,[1] but it can be downloaded from various sites. Other 3270 emulators might be used, but their operation with zPDT must be verified by you. IBM developers have also used recent releases of the IBM PCOMM package (on Microsoft Windows systems).

[1] We were pleased to note that SLES 11 does include x3270.

Do not confuse the levels of the base PC Linux discussed here with the levels of *Linux for System z* that might be run under zPDT.

Base PC hardware

zPDT Version 1 Release 4 has been tested on the following PC hardware:

- Laptops: Lenovo W500 dual core, Lenovo W700 dual core, W510, W520, W530 quad core.
- Servers: IBM xSeries 3500 model 3, IBM xSeries 3650 model 3 and model 4.
- In all cases, a minimum of 8 GB PC memory was available. More general usage is with 8 - 16 GB (or more) memory. Systems with up to 96 GB have been used.
- A CD/DVD drive was present on all test systems, and a USB port used for the zPDT token. (Unpowered USB port extenders should not be used for the zPDT token.)
- Various USB devices, such as disks and flash drives were used to the extent supported by Linux.
- The use of hiperthreading in the base machine is *not acceptable*; hiperthreading (if available on the machine) *must* be disabled at the BIOS level. Also, at the time of writing, the use of “bonded Ethernet interfaces” is not supported.

These are the only *tested* machines for the 1090. Other machines *might* work correctly with zPDT, but they have not been tested. In rare cases, IBM might address zPDT problems only when reported on one of the tested machines.[2] The zPDT system has no specific coded requirements for these particular base machines and operating systems, but the almost infinite number of possible combinations of other hardware and other Linux versions have not been tested.

Functional requirements for a base system

The base machine requirements for the Linux version for zPDT are discussed throughout this book. A summary of the hardware requirements is as follows:

- We suggest PC memory of *at least* 1 GB larger than the intended size of the emulated System z memory.
- Disk space of *at least* 12 GB for Linux (and work space) plus about 2.8 GB for every 3390-3 volume to be emulated. A rather basic z/OS, with few additional products, could be used on a 60-GB disk drive. The tested mobile computers typically had 100+ GB disk drives or larger.
- A suitable USB port must be available for the 1090 or 1091hardware key.[3] Do *not* use an unpowered USB port expander when using zPDT. In particular, do not install the zPDT token in an unpowered USB port expander. (The license server function, described in SG24-7723-05, provides an alternative way to manage the token.)
- A CD/DVD reader usually is needed for loading software.
- Multiple LAN interfaces may be needed in larger configurations, although this is rare.
- Hiperthreading (if available) *should* be disabled at the BIOS level. Hiperthreading can produce extreme slowdowns when z/OS is executing spinloops. If many PC cores are available the slowdowns may be resolved before z/OS console messages are produced, meaning there is no indication of a problem other than reduced performance.
- The Linux distribution must operate correctly on the base PC. New adapters, various power management options, new USB chips, unusual display parameters, new disk

[2] As of the time of writing, this situation had not been encountered.
[3] The use of a remote license server is possible. In this case a USB port is not needed.

technology, and other technology-related items may not work correctly with all Linux distributions or may require additional Linux device drivers or Linux updates.

- The Linux distribution *must* support installation via the `rpm` command. Other software installation management designs do not work with zPDT.
- The basic zPDT offering does not include any System z software. Although System z software may be part of an offering that includes zPDT, the base zPDT product itself does not include any System z software. System z software must be obtained in a media format suitable for a zPDT machine.

System z operating systems

zPDT Version 1 Release 4 has been tested with the following System z operating system levels:

- z/OS 1.11, 1.12, 1.13
- z/VM 5.3, 5.4, 6.1, and 6.2 (There are some limitations to the z/VM 6.2 testing.)
- z/VSE 4.2, 4.3, 5.1
- Linux for System z: SLES 11, RHEL 5.2, RHEL 5.4

Other levels may operate correctly but have not been tested.

2.3 zPDT releases summary

Table 2-1, Table 2-2 on page 21, Table 2-3 on page 21, Table 2-4 on page 22, and Table 2-5 on page 22 indicate key characteristics of zPDT releases. The information under *"Required minimum Linux levels"* indicates the Linux library levels used to compile and link the zPDT modules. Lower level Linux systems should not be used when running the associated zPDT release. Other Linux distributions with libraries at an equivalent level (or later) may be used, although only the other listed distributions were tried by the developers.

Table 2-1 Version 1 Release 4

Characteristic	**Version 1 Release 4**
Date released	December 2012, fixpack May 2013
Initial driver level	45.18
System z level	EC 12 (includes upgrades for z/OS 2.1)
Required minimum Linux level (Earlier levels should not be used)	RHEL 6.1 openSUSE 11.3
Other Linux levels used during development	(openSUSE 11.3, 12.1, 12.2) SLES 11 SP2 (Fedora 15, 17)
Tested z/OS levels	1.13, 1.12
z/VM used during development	6.1, (partial use of 6.2)
Tested z/VSE levels	
Tested Linux for System z level	

Characteristic	Version 1 Release 4
Machines used for testing	Lenovo W520, W530; IBM xSeries 3500-M3, 3650-M3
Virtual environments tested (as of June 2013)	VMWare zBX

Table 2-2 Version 1 Release 3

Characteristic	Version 1 Release 3
Date released	March 2012
Initial driver level	43.20
System z level	z196 (not usable for z/OS 2.1)
Required Linux level (Earlier levels should not be used)	RHEL 5.4 openSUSE 11.2 SLES 11
Informal Linux levels used during development	openSUSE 11.3, 11.4 Fedora 12
Tested z/OS levels	1.13, 1.12, 1.11
z/VM used during development	6.1, 5.4, 5.3
Tested z/VSE levels	5.1, 4.3, 4.2
Tested Linux for System z level	SLES 10, SLES 11, RHEL 5.2, RHEL 5.4
Machines used for testing	Lenovo W520, W530; IBM xSeries 3500-M3, 3650-M3
Virtual environments tested	None

Table 2-3 Version 1 Release 2

Characteristic	Version 1 Release 2
Date released	December 2010
Initial driver level	41.21
System z level	z10, ALS3
Required minimum Linux level (Earlier levels should not be used)	RHEL 5.3 openSUSE 10.3
Informal Linux levels used during development	openSUSE 11.1 Fedora
Tested z/OS levels	1.10, 1.11
z/VM used during development	
Tested z/VSE levels	
Tested Linux for System z level	

Characteristic	Version 1 Release 2
Machines used for testing	Lenovo W500, W510; IBM xSeries 3500-M2, 3650-M2
Virtual environments tested	None

Table 2-4 Version 1 Release 1

Characteristic	Version 1 Release 1
Date released	October 2009
Initial driver level	39.11
System z level	z900, ALS3
Required minimum Linux level (Earlier levels should not be used)	RHEL 5.2 openSUSE 10.3
Informal Linux levels used during development	openSUSE 110.3 Fedora
Tested z/OS levels	1.9, 1.10
z/VM used during development	
Tested z/VSE levels	
Tested Linux for System z level	
Machines used for testing	Lenovo W500, W700, T61p; IBM xSeries 3850
Virtual environments tested	None

Table 2-5 IBM Internal version

Characteristic	Internal version
Date released	July 2007
Initial driver level	45.18
System z level	z800, ALS3
Required Linux level (Earlier levels should not be used)	RHEL 4.4 SuSE 10.1 Remastered
Informal Linux levels used during development	SuSE 10.1 Remastered Fedora
Tested z/OS levels	1.8, 1.9
z/VM used during development	
Tested z/VSE levels	
Tested Linux for System z level	
Machines used for testing	Lenovo T60, T61;
Virtual environments tested	None

2.4 Using older System z architectures

zPDT does not have a facility to emulate older System z architectures. For example, the current release (zPDT Version 1 Release 4) is at the System z EC 12 level. It cannot be set to a System z 196 level or a z10 level. Providing a switchable architectural level facility would result in reduced performance and the product developers are unwilling to make this tradeoff.

If you need to test software under older System z architectures (and older z/OS releases) you must retain older versions of zPDT. Older zPDT releases may or may not work correctly with the latest Linux distributions and IBM cannot provide assistance in this area. In the general case, you must retain older PC hardware, older Linux releases, older z/OS releases, and older zPDT releases if you want to consistently run your software in older operating environments. IBM does not have a way for providing older zPDT releases or older AD-CD releases.

2.5 SCSI adapters

The newest IBM System x servers (at the time of writing) do not list any parallel SCSI adapters for their standard configurations. We understand this to mean the following:

- IBM did not formally test any of the existing SCSI adapters with the newest servers.
- There is no known reason why they should not work.
- We have informally used the UltraSCSI 320 series of adapters with xServer 3650 M2 and 3500 M2 machines without problems with our older SCSI tape drives.

  For some of our systems, we needed to use openSUSE 11.2 or later for this operation. Other and earlier distributions, with Linux kernels below the level used in openSUSE 11.2, did not work with these SCSI adapters on some of our systems. This condition is likely to change with future Linux distributions. If parallel SCSI operation is important to you, we *strongly advise* that you discuss your zPDT configuration with your zPDT provider.
- There is no *defined* IBM support for these configurations.
- Parallel SCSI adapters, cables, and devices can be complex. There are different data path widths, single-ended and differential circuits, low-voltage and high-voltage versions, and a variety of terminators. If you are not familiar with this area, we *strongly suggest* you obtain expert help in configuring your system.
- The newest SCSI devices use fiber connections instead of parallel (wire) connections.

SCSI tape drives

Some SCSI tape drives may be used with zPDT. They can be used via Linux utility functions or used directly by the System z operating system (where they appear to be IBM 3490[4] drives). Not all SCSI tape drives are usable by zPDT. The usability depends on the exact model, the exact firmware level, the exact SCSI adapter used, and the firmware options that are set in the drive. IBM has used a variety of different SCSI drives for testing, but IBM cannot predict whether *your* SCSI drive will work with zPDT. If this is important to you, we *strongly suggest* that you discuss your requirements with your zPDT provider.

More information about the SCSI drives tested by IBM is provided in the third book in this series (SG24-7723-02 and later).

[4] This also means they are limited by IBM 3490 architecture controls.

3

zPDT components

At the highest level, zPDT has or needs the following components:

- A base Linux system.

  This is not provided with zPDT. The user must acquire this directly.
- A suitable 3270 emulator (which is usually run on the same personal computer that is hosting zPDT, although this is not required).

  At least one 3270 emulator (x3270) is provided with some Linux distributions, but not with others. Other modern 3270 emulators might be used, but verification of their operation with zPDT is up to the user. The zPDT package does not provide a 3270 emulator.
- The 1090 or 1091 hardware USB token, which is required for zPDT operation.[1]
- The zPDT program file.[2] Within this file are the following:
  - Two prerequisite programs for communicating with the 1090 or 1091 token. These two packages are provided with zPDT and only these provided versions may be used; other versions available from the web *should not be used*. (A remote license server may be used instead of the local token.)
  - The Red Hat (RHEL, Fedora) version of zPDT.
  - The Novell (SLES, openSUSE) version of zPDT.
  - An installer program that displays a license, installs the prerequisite programs (if not already present), and then selects and installs the correct zPDT version.
  - Components that provide remote license and identity management functions.
- System z software, such as z/OS, is not part of zPDT. It must be licensed and acquired separately.

The remainder of this chapter discusses the components in the zPDT rpm[3] (after it is installed). The discussion is the same whether the Red Hat or Novell versions are used and whether the 1090 or 1091 package is used.

[1] The general RDz license manager may be used in addition to1091 keys.

[2] There are two program file packages, one for 1090 systems and one for 1091 systems.

[3] zPDT is installed via the rpm function, but cannot be installed directly with the **rpm** command. The zPDT installer program must be used. This is discussed in the second document in this series (SG24-7722).

3.1 zPDT elements

The executable elements of the zPDT package (normally placed in `/usr/z1090/bin` on the underlying Linux system) are in three general categories:

- System z operation, which is provided by a primary zPDT program module and a number of associated DLL modules.
- Several device emulation modules, known as *device managers*.
- Multiple command modules to configure, start, stop, and manage zPDT operation. These are executed as simple Linux commands, working from a Linux terminal window.

zPDT installation and use also creates `/usr/z1090/man` and `/usr/z1090/uim` directories. The `uim` directory may contain two small files that are used to provide a consistent serial number for System z. The `man` directory contains normal Linux man pages.

The installation and use of the z1090 rpm,[4] in addition to loading executable modules in `/usr/z1090/bin`, creates a number of subdirectories (placed in the z1090 subdirectory) in the user's home directory.[5] Briefly, these subdirectories are:

- *cards*, *lists* - May be used to provide input files to an emulated card reader or output from an emulated printer. If not used, they are empty.
- *disks*, *tapes* - May be used to hold emulated disk or tape volumes, but these subdirectories are typically not used for anything. The emulated volumes are usually placed elsewhere, in other Linux file systems.
- *logs* - Used by zPDT to hold various dumps, logs, and traces. zPDT partly manages the contents of this subdirectory. The contents of this directory are important if it becomes necessary to investigate a zPDT failure.
- *configs*, *pipes*, *srdis* - Used for zPDT internal processing; do not erase or alter the contents of these small subdirectories.

Finally, a device map (devmap) is needed for zPDT operation. This element is not provided by zPDT, but must be created by the user.

The System z operational modules are not further described. They are not directly used or referenced by the zPDT user. The device managers are described in 4.3, "Manager stanzas" on page 38. The syntax of the zPDT commands is described in 4.4, "zPDT commands" on page 49. Practical uses of zPDT commands, device managers, and devmaps are explained, at length, in Volume 2: *Installation and Basic Use,* and Volume 3: *Additional Topics* of this documentation series.

3.1.1 Memory

The complete zPDT environment exists in Linux virtual memory. Linux is aggressive in allocating real memory frames to virtual memory pages and disk file data using its own (Linux) judgment about what is the best use of real memory. Starting another application (especially a graphics application) can consume much virtual memory in the new process.[6] This can cause page stealing from other processes to supply real memory frames. The situation is more complex when Linux caching of disk I/O is considered, and disk caching is a very important element of Linux operation and performance.

[4] The z1091 rpm also installs in /usr/z1090/bin.

[5] When zPDT is started, a z1090 subdirectory is created in the home directory of the user (if it does not already exist). The subdirectories discussed here are under the z1090 subdirectory.

[6] This might not be in shared storage and may not affect the maximum size of emulated z/OS memory.

We consider 4 GB of memory (in a PC) to be the absolute minimum for zPDT usage. A 4 GB machine is usable for a modest z/OS system. Memory may be much larger. For example, one of the zPDT test environments uses a 96 GB PC and runs multiple 16-64 GB z/OS images.

It is important to understand that zPDT simply exists in Linux virtual memory. We might informally say something like, “With a 4 GB machine we can allocate 1 GB to Linux and 3 GB to zPDT,” but such statements must not be taken literally. zPDT does not physically partition PC memory in any way. If we inspected the machine in this example at a random time, we might find 1.2 GB owned by the primary zPDT module, 0.2 GB owned by recognizable core Linux functions, 1.8 GB used for disk data cache, 0.2 used by various other processes (such as zPDT device managers and so forth) and the rest unassigned. A few seconds later, the usage statistics might be different.

We suggest that the PC memory size be *at least* 1 GB larger than the sum of all concurrent zPDT-defined System z memory. More is better because it allows the Linux disk cache to perform better.

The primary goals are to (1) avoid Linux paging that stalls zPDT operation, and (2) to allow Linux to have an effective disk cache. There is no easy way to directly manage either of these goals. They are indirectly managed by providing ample PC memory.

3.1.2 Disk space

The disk space for the 1090 executable programs and control files is relatively small.[7] The disk space for emulated System z volumes is not small and therefore some planning is needed. The space for emulated disk volumes may be calculated accurately, while the space for emulated tape volumes depends completely on the amount of data on the emulated tape volumes.

For practical purposes, we consider only 3390 emulated disk volumes. For the standard 3390 models the approximate required space is as follows:

3390 model	**Approximate space required**	**Exact space required**
3390-1	.95 GB	948,810,752 bytes
3390-2	1.9 GB	1,897,620,992 bytes
3390-3	2.8 GB	2,846,431,232 bytes
3390-9	8.5 GB	8,539,292,672 bytes
1 3390 cylinder		852,480[8]

The *per cylinder* space may be used to calculate the disk space needed for nonstandard 3390 sizes.

Tape sizes reflect the size of the data written on the tape with a very small additional space (less than 1%) needed for awstape control blocks.[9] (Optionally, the awstape device manager can compress these files, often greatly reducing the amount of space used.)

3.1.3 LAN adapters

We consider only Ethernet adapters in this discussion.[10] A Linux-based zPDT system can use more than one LAN adapter, although this is unusual. We must consider several “users” of LAN adapters in the base machine:

[7] It is typically less than 30 MB.
[8] Each volume has an additional 512 bytes overhead.
[9] The actual overhead is 6 bytes for each block written (including a tape mark, which counts as a block).
[10] Wireless adapters are also Ethernet adapters.

- Linux itself is normally a LAN user. Remember that the emulated local 3270 connections (via the aws3274 device manager) are connected through Linux TCP/IP.[11]
- z/OS (or z/VM, or z/VSE) TCP/IP, if used, needs a LAN adapter. This usage may be in one of two different modes:
  - Non-QDIO mode, in which an older IBM 3172 control unit (or LAN Channel Station, LCS) is emulated.
  - In QDIO mode, which is recommended.
- z/OS (or another operating system) might use a LAN for SNA connections, although this is not tested or supported by IBM. This requires non-QDIO mode.

A LAN adapter may be shared between zPDT OSA and the base Linux system with the following rules and restrictions:

- A given LAN adapter may be used for OSA Express emulation in either QDIO or non-QDIO mode, but not both. The selection of QDIO or non-QDIO is made in the devmap definitions; the awsosa device manager is used in both cases.
- Adapter sharing between OSA and the base Linux system is independent of whether QDIO or non-QDIO mode is used for OSA.
- A given adapter may be used by both OSA (either mode) and base Linux connections. For example you can use Linux telnet, ftp, Web browser (or server), the aws3270 device manager, and so forth at the same time that OSA is using the same Ethernet adapter.
- A logical connection between Linux TCP/IP and OSA TCP/IP can be made only by using an intermediate virtual interface (which we describe as a *tunnel*).
- Remember that the aws3274 device manager (which accepts TN3270e clients and emulates local, channel-attached 3270 devices) does not use OSA.

Wireless LAN

Wireless LAN connections may be used with the 1090, but there are considerations involved:

- Wireless usage almost always involves DHCP. Standard z/OS is not a DHCP client. This means the wireless functions are between a remote client and Linux. In practice, this means they are used with 3270 emulators connected to the aws3274 device manager. The MVS console and up to 31 TSO users may be connected this way.
- Temporarily dropping a link is common with wireless connections and usually has minor effects for typical mobile computer users. Dropping a link that runs the MVS™ console, for example, produces more than a minor effect. Some Linux wireless environments allow considerable time (many seconds) for a dropped wireless connection to reconnect. This can create unexpected timeouts for z/OS functions, depending on the exact state of the system when the connection drop happened.

Informally, we have found wireless connections practical when used in the same room (such as a classroom) where dropped connections are very unlikely.

3.2 Device managers

zPDT provides 12 device managers:

- aws3215 - Emulates a 3215 console device (seldom used today), using a Linux terminal window for the interface.

[11] If local x3270 windows are the only TCP/IP functions used under the base Linux, then the *localhost* connection (127.0.0.1) can be used and this does not tie up a hardware LAN adapter.

- aws3274 - Emulates a local, channel-attached 3274 control unit. This device manager is almost always used to provide the MVS console, for example, and 3270 application sessions. TN3270 sessions are used, via the base Linux TCP/IP interface.
- awsckd - Emulates 3390 (and 3380) disk units, using a Linux file for each 3390/3380 device.
- awscmd - Emulates a 3480 tape drive, but routes output records to the base Linux system where they are executed as commands, and returns Linux output to the emulated tape drive.
- awsfba - Emulates FBA devices, which are supported by z/VSE and z/VM. A Linux file is used for each emulated device.
- awsoma - Emulates the Optical Media Attach interface, working with Linux files in this format. (This is mostly of historical interest, and is seldom used today.)
- awsosa - Emulates most functions of an OSA-Express2 Ethernet interface, and is used by TCP/IP (in either OSE or OSD modes) and by SNA[12] (in OSE mode).
- awsprt - Emulates a 1403 or 3211 printer, using a Linux file for output.
- awsrdr - Emulates a 2540 card reader, using Linux files as input. (The 2540 card punch functions are not emulated.)
- awsscsi - Uses a SCSI-attached tape drive to emulate a 3490 tape drive, providing a way to read/write “real” mainframe tape volumes.
- awstape - Emulates a 3420/3480/3490/3590 tape drive, using a Linux file as the tape media.
- awsctc - Emulates an IBM 3088channel-to-channel adapter using TCP/IP as the communication mechanism.

A typical zPDT user, running z/OS, would normally use aws3274, awsckd, awsosa (if connectivity other than local 3270s is needed), and perhaps awstape. The other device managers are less often used.

3.3 Device maps

A device map, commonly known as a devmap, is a simple Linux text file. You may have many devmaps, each a separate Linux file. One devmap is specified when zPDT is started; you can use a different devmap each time a zPDT instance is started. A devmap specifies the System z characteristics to be used and the device managers (with their parameters) to be used for an instance of zPDT operation.

3.4 Directory structure

The 1090 has the following *default* directory[13] structure in Linux:

```
Directory path                     Purpose
/home/<userid>/z1090/logs/         various traces are placed here
/home/<userid>/z1090/configs/      (internal 1090 functions)
/home/<userid>/z1090/disks/        emulated disk volumes
/home/<userid>/z1090/tapes/        emulated tape volumes
/home/<userid>/z1090/cards/        input to the emulated card reader
/home/<userid>/z1090/lists/        emulated printer output
```

[12] SNA usage is not supported by IBM at this time.

[13] These names are subject to the discussion about the home directory. You should substitute the appropriate home directory name for the */home/<userid>* portions of these names. A home directory could be almost anywhere in the root file system or in another file system. Our examples are based on the default form used by current Linux distributions.

```
/home/<userid>/z1090/pipes/          (internal 1090 functions)
/home/<userid>/z1090/srdis/          (internal 1090 functions)

/usr/z1090/bin                       executable 1090 code, scripts
/usr/z1090/man                       minor documentation
/usr/z1090/uim                       identity manager files
```

Notice that different userids would have different default 1090 directories and files. 1090 operation is sensitive to the Linux userid being used. The use of the default logs, lists, and configs directories is mandatory for some operations, but is optional for other files such as emulated disk and tape volumes. Emulated devices have default *file* names, based on the assigned device number, but may use specified file names instead of the default file names. (We always use specified file names in our examples. None of our examples use the default disk and tape subdirectories and these are typically empty.)

These subdirectories are created in the current home directory (if they do not already exist) when zPDT operation is first started.

Using the default directory paths and file names (for emulated volumes) provides a simplified startup process because less information needs to be provided in the control files and the starting command. It also makes the emulated volumes potentially private to a particular userid.

However, using the default directory paths and file names for emulated disk and tape volumes has several disadvantages:

- The /home directory is typically part of one of the base Linux file systems. If Linux is reinstalled, for example, the /home file system might be lost. Accidental loss is minimized if a separate file system is maintained for emulated volumes. An installation may have much effort invested in System z work on their emulated disk volumes and want to protect these as much as possible.
- The default 1090 naming convention ties emulated volume file names to emulated addresses. This can produce conflicting names if multiple versions of an operating system are installed, especially when the operating system uses *well-known* addresses for various functions.
- We may not want access to emulated disk and tape volumes to be sensitive to a particular Linux userid.
- Sharing devices (“shared DASD”) between multiple zPDT instances does not work when using default directory and file names for emulated volumes.

The remainder of this document ignores the use of default file names for emulated I/O devices. We recommend using a separate (and large) Linux file system for emulated volumes. This insulates them from Linux reinstallations and also insulates both the emulated volume file system and the base Linux file system(s) from unplanned growth in each other.

For these reasons most of the examples in this book assume that all emulated I/O files are placed in the /z directory.[14] In our case, when we installed Linux, we created a separate partition (with a large amount of disk space) that is mounted at /z. We use this to hold all the emulated volumes. The `cards` and `lists` directories, in the default directory path, are seldom used in typical operation and we elected to use the default paths to these files.

[14] The mount point name, /z in our examples, is completely arbitrary.

3.5 1090 control structure

The general structure of 1090 control files is shown in Figure 3-1.

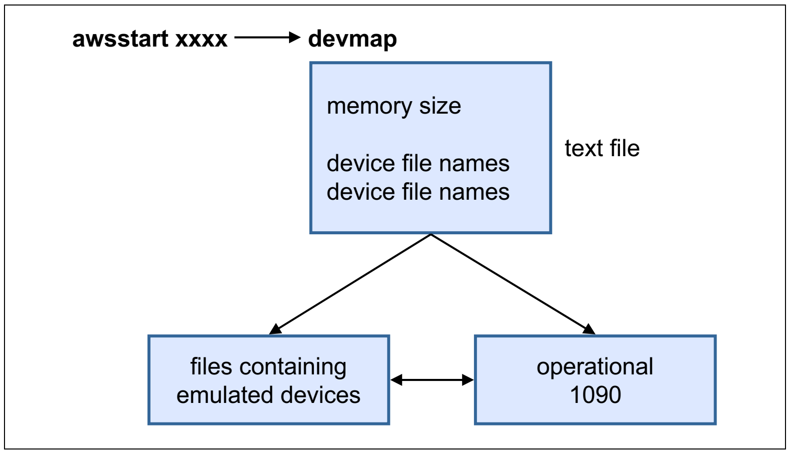

Figure 3-1 Control files - general structure

A 1090 System z machine is started with the **`awsstart`** command, issued from a Linux terminal window. A parameter of this command points to a device map or *devmap*. This is a simple Linux text file containing specifications for the System z machine.

4

Reference

In this chapter we provide reference information for zPDT device map entries (for device managers) and for zPDT commands. Information and guidance for using these device managers and commands is found throughout the zPDT set of IBM Redbooks publications.

4.1 Device maps

A device map (devmap) consists of a system stanza, an optional adjunct processor stanza, and a variable number of device manager stanzas. The descriptions in this chapter are intended to provide syntax and format information, but are not intended to represent typical usage. Usage information is provided in other volumes of this series of books.

A device map is a simple Linux text file with an arbitrary file name. Many devmaps may exist, but only one can be in use for an instance of zPDT. It is possible to have multiple zPDT instances running, each under a different Linux userid. Each instance has its own devmap. The use of multiple zPDT instances is discussed in Volume 3 of this series of books. The remainder of this chapter assumes a single instance of zPDT is being used.

Devmap files should be entered in lower case,[1] except for parameters that specify Linux file names. Devmap statements begin in the first column of each statement. Stanzas are separated by blank lines. A hash or pound sign (#) signals the beginning of comments. The square brackets shown in the descriptions below are part of the syntax and must be entered as shown.

zPDT reads the specified device map when various components of zPDT are started. It does not process updates to the devmap while zPDT is running. To alter the operational device map, zPDT must be stopped and then started again with the new or revised devmap. However, the Linux file associated with some devices (such as emulated tape drives or emulated disk drives) can be dynamically changed while zPDT is operational by using the **awsmount** command.

4.2 System stanza

A [system] stanza might look like this:

```
[system]
memory 2500m              # define 2.5 GB memory for System z
processors 1              # this defaults to 1; maximum is 3 for an L03 model
3270port 3270             # specify unique IP port number for aws3274
expand 0m                 # no expanded storage
int3270port 3271          # HMC integrated 3270 function
ipl 0A80 “0A8200”         # automatic ipl control (optional; not recommended)
cpuopt alr=on             # optional function (defaults to on)
command 2 x3270 localhost:3270
```

The `memory` statement specifies the size of the System z memory to be used for zPDT operation. For performance reasons the real memory size of the PC should be *at least* 1000 MB greater than the `memory` parameter.[2] The number specified must be smaller than the maximum shared memory value specified for Linux; this is set by the kernel.shmmax parameter in Linux.[3]

The `processors` statement specifies the number of System z CPs to be used in this instance. The default is one. This number must not be more than the zPDT token model allows, and

[1] This is not required by some elements of a devmap, which ignore upper/lower case differences. Not all devmap elements do this. To avoid problems, we recommend using lower case for everything (except for Linux file names, which are case sensitive).

[2] This statement assumes a simple, dedicated environment. Other environments may require more planning for effective memory use.

[3] This kernel variable is specified by the instructions in Volume 2 of this series of books.

should be less than the number of real processors in the base computer. The `processors` statement is also used to indicate the use of speciality PUs, as in the following examples (where “cp” indicates a normal, general-purpose CP):

```
processors 3               # three CPs. Assumed “cp” type
processors 3 cp cp ziip    # two CPs and one zIIP
processors 2 cp cp zaap    # invalid. Two processors, but three definitions
processors 1 ziip          # invalid. Must have at least one cp
processors 3 cp zaap ifl   # one of each
processors 3 zaap cp cp    # invalid; cp must be first in the list
```

The operands for the `processors` statement are the number of processors (typically 1, 2, or 3)[4], which cannot exceed the number allowed by the zPDT token model number. The processors default to CPs; if speciality processors are wanted, these should be listed after the number as shown in the examples. The processor types are cp, ziip, zaap, and ifl. If zIIPs or zAAPs are specified in the processors statement, at least one cp must be listed first.[5]

Note that z/VM can simulate zIIP and zAAP processors, using normal CPs for the simulation. Using z/VM for this function can reduce the number of zPDT licenses needed by the total system.

The `expand` statement specifies the size of expanded storage for the System z machine. This is optional. z/OS no longer uses expanded storage. However, z/VM still uses it. There is no particular upper limit when working under 64-bit Linux systems, although the size should usually be less than the physical memory size of the base machine.

The `3270port` statement specifies a port number to be used by Linux TCP/IP for the aws3274 device manager. This must be an unused port and is typically a number greater than 1024. We arbitrarily use port 3270 because it is easy to remember. A TN3270e connection to this Linux port appears as a local, channel-attached 3270 to the System z.

The *ipl* statement is optional and indicates that the **`ipl`** command is to be executed automatically when the zPDT operation is started. However, using this option might prevent you from connecting 3270 emulator sessions at an appropriate time. We suggest not using this option.

The `cpuopt` statement specifies optional parameters for the CPs. The only valid parameters at this time are these:

```
cpuopt asn_lx_reuse=on              (no blanks in operand)
cpuopt asn_lx_reuse=off             (no blanks in operand)
cpuopt zVM_CouplingFacility         (no blanks in operand)
cpuopt alr=on,zVM_Coupling          (abbreviations)
```

The *asn_lx_reuse* operand may be abbreviated as *alr*. The *zVM_CouplingFacility* operand may be abbreviated as *zVM_CouplingFac* or *zVM_Coupling*.

The asn_lx_reuse parameter (which may be abbreviated as alr) defaults to “on” and this is the normal mode of operation for zPDT. This mode matches the relevant architecture of IBM z10 and later machines. When this parameter is set to “off” zPDT indicates that the LX and ASN REUSE facility is not present. This mode *may* be useful for running early z/OS releases. The use of “alr=off” produces an environment that is not supported or tested by IBM. While it may be useful for working with earlier z/OS releases, the user must assume all responsibility for correctness of operation and the correctness of results. Note that there are no blanks in these operands; there must not be a blank before or after the equal sign.

[4] The number of processors for an instance has a maximum value of 8. This is usable only if multiple tokens are used or nonstandard “large” tokens are used, and may be limited to zPDT license terms and conditions.
[5] The first processor type listed becomes the IPL processor; zIIPs and zAAPs cannot handle an IPL.

The zVM_CouplingFacility operand is significant only for 1091 systems, which must have the proper license feature to enable it. In effect, the zVM_CouplingFacility function is always present for 1090 systems.

The `command` statement specifies Linux commands that may be automatically executed as part of the zPDT operation. The syntax is:

```
command phase-number [synchronous] command-string
```

The phase number is a digit from 1 to 4:

- Phase 1 means the command is to be executed before zPDT is started.
- Phase 2 means the command is to be executed after zPDT is initialized.
- Phase 3 means the command is to be executed just before zPDT is shut down.
- Phase 4 means the command is to be executed after zPDT is shut down.

By default, commands are executed asynchronously but may be forced to synchronous operation. (This should seldom be used, since it forces other zPDT operations to wait until the command is completed. For example, do not use it for x3270 startup.) The word *`command`* may be abbreviated to *`cmd`* and *`synchronous`* may be abbreviated to *`sync`*. If asynchronous commands terminate while zPDT is still running, they are not restarted. If they are still running when zPDT is shut down they are sent a SIGTERM signal and should terminate.

Additional system stanza options include the following:

```
[system]
...
rdtserver 27000@our.server.acme.com    # RDz license server and port
int3270port 3271                       # HMC-style integrated port
intASCIIport 3300                      # HMC-style ASCII port
```

The `rdtserver` statement is used only with a 1091 system. It points[6] to an RDz license server used to supplement the 1091 token.[7] Note that an RDz license server is not the same as a zPDT remote license server. The operand may be a normal URL domain name or an absolute IP numeric address.

The int3270port and intASCIIport statements provide emulation for HMC-style integrated terminal functions. The operand for each statement is a port number on the local Linux system. After starting zPDT with one (or both) of these operands you would start a 3270 emulator connected to the indicated port number or start z1090term[8] connected to the indicated ASCII terminal port number. These emulated terminals need not be on the base Linux system.

A reasonable example of a system stanza could be as follows:

```
[system]
memory 6000m
processors 2
3270port 3270
command 2 x3270 -model 4 -geometry +100+100  localhost:3270
command 2 x3270 -model 4 -geometry +2100+100 localhost:3270
command 2 sync ipl a80 parm 0a8200
command 4 echo 'zPDT operation has completed'
```

An ampersand (&) is not needed after the x3270 commands in a [system] stanza. The geometry parameters are optional, of course. Simply place the x3270 panels at convenient

[6] Note the syntax: port@ipaddress. There is nothing special about the port number (27000) used in the example.
[7] Contact your IBM marketing representative for more information about RDz general licenses.
[8] See the command descriptions in Chapter 4.

places on the Linux desktop. Also, the x3270 sessions are automatically closed when zPDT is shut down.

A devmap has several additional features.[9] These are:

- The use of Linux environmental variables
- The **include** function
- The **message** function

An example of each of these functions is included in the following devmap:

```
[system]
memory $(SIZE)
3270port 3270

[manager]
name aws3274 1234
device 0700 3279 3274
device 0701 3279 3274

include dasd.def

[manager]
name awsosa 4567
device 0400 osa osa
...
message Remember to start or connect the x3270 sessions before you IPL
message
message For normal startup ipl A80 parm 0a8200
```

This devmap references a second file, `dasd.def` (in the same directory), which might contain:

```
[manager]
name awsckd ABCD
device A80 3390 3990 /z/SBRES1
etc
```

The SIZE parameter in this example is a Linux environmental variable. The variable name must be enclosed in parenthesis, as shown. The value of the variable must be set before the devmap is used. It can be set by the Linux shell command, for example:

```
$ export SIZE=3500m
```

This command can be issued prior to an **awsstart** command (in the same Linux terminal window), but then it will not be effective in other Linux terminal windows. zPDT commands that reference the active devmap (such as **awsstat**) could not be used in other terminal windows. Alternatively, the **export** command can be added to the `.bashrc` file, where it will be effective for any terminal window subsequently opened. For practical purposes, we suggest adding any devmap environmental variables to the `.bashrc` file.[10] If the specified environmental variable is not set, a null string is placed in the devmap.

The **include** function in the example operates as might be expected. The file specified is logically inserted into the devmap at the point shown. The operand of the **include** function can specify a full Linux path name; if a simple name is specified, it is assumed to be in the current directory. The file name specified cannot contain blanks. The name could be an

[9] These features were added for special purposes. We have not seen much usage by typical zPDT users.
[10] Other required changes to the `.bashrc` file are described in the second volume in this series, SG24-7722.

environmental variable instead of a file name, for example **include $(fileVAR)**. If the specified environmental variable is not defined, the **include** function is skipped.

The **message** function simply displays its text when the devmap is processed by the **awsstart** command. The *message* function name can be abbreviated to *msg*.

It is very unlikely that all the [system] stanza options would be used at one time, but here is a full example for reference:

```
[system]
memory 6000m
processors 3 cp cp ziip
3270port 3270
int3270port 3271
intASCIIport 4000
rdtserver 6700@192.168.1.200
expand 1000m
#ipl 0A80 “0A8200”      (This statement is not recommended)
cpuopt alr=on,zVM_Coupling
message This devmap is excessive
command 2 x3270 localhost:3270
command 2 x3270 -geometry +1100+100 localhost:3270
command 2 x3270 -geomentry +1100+600 localhost:3271
command 2 sync ipl a80 parm a08200
message Remember to start more 3270 sessions
include devmap2
```

4.2.1 Adjunct-processor stanza

The zPDT system provides emulation of the System z cryptographic adapter.[11] The basic devmap format is as follows:

```
[adjunct-processors]
crypto 0
crypto 1
```

This defines two cryptographic processors, numbered 0 and 1. If multiple zPDT instances and shared cryptographic processors are used, the sharing instances may have a definition such as the following:

```
[adjunct-processors]
domain 0 2
domain 1 2
```

This indicates that the instance is using domain 2 in cryptographic coprocessors 0 and 1. See the third document in this series (SG24-7723) for more details.

4.3 Manager stanzas

A device manager stanza has the following general format:

```
[manager]
name awsckd C700
device 0a80 3390 3990 /z/SBRES1
```

[11] Do not confuse this with the cryptographic instructions, which to do not require any special devmap statements.

```
device 0a81 3390 3990 /z/SBRES2
etc
```

The stanza begins with [manager], including the square brackets. In this example the device manager name is awsckd, but this could be any of the supported device managers. The device manager name is followed by an arbitrary hex number (up to four digits, different for each name statement)[12]. The *name* statement is followed by as many *device* statements as needed. The general format is:

- For name statements:
  - The constant “name” starting in the first column.
  - The device manager name, such as awsckd.
  - A hex control unit number; each name statement must have a different number.
  - Additional optional parameters, such as:
    - --path=xx to specify an emulated CHPID number.
    - --pathtype=xxx to specify an emulated CHPID type (usually EIO).
    - --compress to specify compressed awstape generation.
    - various optional *tunnel* parameters for OSA operation.
- For device statements:
  - The constant “device” starting in the first column.
  - The device number (“address”) to be used, expressed in hexadecimal. This may be three or four digits.
  - The device type, such as 3390. This must specify a correct device type for the device manager.
  - The control unit type associated with the device. (This parameter is not used for anything at this time, but it would be unwise to not use an appropriate control unit number.)
  - Parameter(s) unique to the device:
    - A fully qualified file name.
    - --unitadd=x, to specify a unit address (as it would appear in an IOCDS) for some device types (such as OSA). If this parameter is not used, the two low-order digits of the device number are used as the default unit address for OSA devices. The default is appropriate in almost all cases.
    - Additional parameters for OSA operation.

The --path, --pathtype, and --unitadd parameters are typically used only for OSA definitions.

Except for OSA devices, the path for emulated devices defaults to 01 and the pathtype defaults to EIO.[13] In very rare cases it may be desirable to change these values. This can be done with the --path and --pathtype operands on a name statement:

```
[manager]
name awsckd 20 --path=30 --pathtype=eio
device A90 3390 3990 /z/specialvolume
```

The path value is expressed as a hex number. Multiple stanzas for the same device manager may be used. A maximum of 255 devices may be listed in a stanza, where multiple devices are not limited by limitations of the emulated control unit. The device numbers (addresses) assigned to each device need not be sequential or in any particular order.

[12] This parameter originally matched a number in a separate IOCDS file. This separate IOCDS is no longer used, but the positional parameter in the *name* statement remains.

[13] EIO is a special CHPID type for Emulated I/O. zPDT users do not normally need to specify this anywhere.

4.3.1 The awsckd device manager

The awsckd device manager emulates 3380 or 3390 disk drives. The definitions for awsckd are simple, as this example illustrates:

```
[manager]
name awsckd 4321
device a80 3390 3990 /z/SYSRES
device a85 3390 3990 /tmp/my3390vol
device aa7 3390 3990 /z/SARES1
etc
```

The device type can be 3390 or 3380; in either case, the Linux file named by the fourth parameter of device statements must be in the appropriate emulated format for that device type. The Linux file containing the emulated volume must have been created with the **alcckd** command, or copied from media that originated on a system where the file was initially created with **alcckd**. Each emulated volume is a single, separate Linux file.

The most common CKD devices are 3390 units. Standard 3390s (models -1, -2, -3, and -9) can be used, or a variable number of cylinders can be used. The maximum size for a normal 3390 is 64K-1 cylinders; however, zPDT can provide Extended Address Volume (EAV) 3390s, as well.[14]

The “extra” cylinders of a 3390 are not emulated; these are the cylinders reserved as spares or for diagnostic use. For example, a 3390-3 contains 3339 usable cylinders, and this is what is emulated. Parallel access to volumes (PAV) is not supported.

4.3.2 The awsfba device manager

The awsfba device manager provides emulation for FBA disk devices (as used by z/VM and VSE).

```
[manager]
name awsfba 6543
device 100 9336 9336 /z/DOSRES
device 101 9336 9336 /z/DOSWRK
```

awsfba devices (volumes) must be created before they can be used. This is done with the **alcfba** utility.

4.3.3 The aws3274 device manager

The aws3274 device manager emulates local, channel-attached, non-SNA 3270 sessions. These are used for MVS consoles, simple VTAM® sessions (TSO, CICS®, and so forth), z/VM terminals, and similar purposes. The actual 3270 emulators (x3270, PCOMM, or other 3270 emulators) might be local (on the underlying Linux system running zPDT) or remotely connected via a TCP/IP connection to the underlying Linux. In either case they use the Linux TCP/IP port number that is assigned in the [system] section of the devmap and they appear to be local, channel-attached 3270s to the System z software. The same Ethernet interface can be used for Linux functions, such as telnet, aws3274, ftp, and so forth and also for OSA connections.

There is a maximum of 32 emulated local 3270 device sessions, regardless of the number of aws3274 stanzas.

[14] A “large volume” 3390 has more than 64 K cylinders. Usage is being introduced with z/OS 1.11.

The devmap parameters for emulated local 3270s offer a number of options. These are best explained by an example.

```
[manager]
name aws3274  C700                    # C700 is an arbitrary CUNUMBR
device 0701 3279 3274 L701
device 0702 3279 3274 L702
device 0703 3279 3274 TSO
device 0704 3279 3274 TSO
device 0705 3279 3274 TSO
device 0706 3279 3274
device 0707 3279 3274
device 0708 3279 3274 IMS
device 0709 3279 3274 IMS
device 070A 3279 3274 IMS
device 070B 3279 3274 IMS
device 070C 3279 3274 L70C
device 070D 3279 3274
device 070E 3279 3274
```

The three operands after the *device* keyword are the address (device number), the device type, and the control unit type. The remaining operand controls potential TN3270e client connections to the device. This operand is known as an LUname, although it is not used as a real SNA LU name. (TN3270e clients can pass an LUname, intended for SNA protocols, during startup. We use this LU name passing facility here, without actually passing it to VTAM.)

In this example, LUnames L701, L702, TSO, IMS™, and L70C are used. The connection rules are:

- The LUname is not case sensitive.
- If an LUname is specified by the TN3270e client, then a free device with the matching LUname will be used.
- If no LUname is specified by the TN3270E client, the next free device in the list is used.
- If there is no free device to match the specified LUname, the connection is rejected.
- A device is freed when a previous TN3270E client connection is terminated.
- If no LUname is specified in the devmap, the default LUname Dev-nnn is generated, where nnnn is the device address.
- Up to 32 TN3270e clients may be connected to this device manager.

The aws3274 device manager listens on a port in the base Linux TCP/IP system. Assume the Linux TCP/IP address is 192.168.0.40 in the following examples. Also assume that our devmap specifies 3270 as the aws3270 port number. A user could enter one of the following commands to establish an x3270 session:

```
$ x3270 -port 3270 192.168.0.40 &           case one
$ x3270 -port 3270 TSO@192.168.0.40 &       case two
$ x3270 -port 3270 L702@192.168.0.40 &      case three
$ x3270 -port 3270 IMS@localhost &          use local system
```

Assume our x3270 client is on a remote machine connected to a private LAN that includes the zPDT system. In case 1, the user is connected to the next available 3270 session (in the devmap list). In case 2, the client is connected to the next free device with LUname TSO. In case 3, the client is connected to the single device with LUname L702, provided that device is free at this time. The fourth example illustrates that the same LUname rules apply to connections from the Linux desktop.

In this example both TSO and L702 are LUnames. TSO happens to be used multiple times but L702 is used only once. There is no requirement to have this arrangement and no requirement to have the LUname reflect the device address (device number).

The devmap for an AD-CD z/OS system might be defined like this:

```
[manager]
name aws3274 C700
device 0700 3279 3174
device 0701 3279 3174
device 0702 3279 3174
device 0703 3279 3174
.....
device 070A 3279 3174
```

Connections take the next free terminal in the devmap list if no LU conditions are specified. This can be useful if the first terminal in the devmap is the MVS console[15] and the next terminal is a suitable TSO address. In this case, without specifying any LU names, the first x3270 session will be the MVS console and the second will be a TSO session (or CICS or some other VTAM application).

From the user's perspective, each 3270 terminal is a TN3270e session. The IBM Personal Communications product and the x3270 emulator provided with many Linux distributions have been tested for this usage.[16] The TN3270e client might operate on the machine running the zPDT processes (on the local Linux graphic panel, for example), or it might operate through a remote TCP/IP connection. In either case, the TN3270E terminal appears as a local, non-SNA, channel-attached 3270 to the System z operating system.

The use of TN3270e (instead of TN3270) is required because the LU name (which is supported by TN3270e, but not TN3270) is needed. Most modern, supported 3270 emulators provide TN3270e functions.

4.3.4 The awstape device manager

Definitions for awstape appear as follows:

```
[manager]
name awstape AB00 --maxlength=1000m
device 560 3490 3490
device 561 3490 3490 /local/my.tape.vol.111111
```

The emulated device type may be 3420, 3480, 3490, or 3590. (The third operand, the control unit type, is not meaningful.) A file name may be specified as the last operand; if a file name is specified, the file must be in awstape format (if it is for input). This situation is similar to a premounted tape on a larger System z. Typically, no file is specified for emulated tape devices. Instead, the **awsmount** command is used to emulate the mounting of a tape volume.

The `maxlength` parameter is optional. If a `maxlength` value is specified, the device manager signals end-of-tape after the specified amount has been written. (z/OS would then probably write trailer labels and call for another tape mount.) If `maxlength` is not specified, then the maximum tape length is limited by one or more of the following conditions:

[15] The AD-CD z/OS systems have always defined the MVS console at address 700.

[16] The aws3274 device manager sends an attention signal to the host when a session is first connected. In some cases, such as when connected to the VTAM unformatted system services function, this may prompt a full buffer read by the host software. If the TN3270e session buffer is not formatted for this buffer read, the host may display an "Unsupported Function" message. Simply clearing the TN3270 screen should resolve the situation. Some TN3270e emulators encounter this situation and others do not.

- The amount of free disk space in the Linux file system.
- An architectural limit of approximately four million tape blocks for 3480 and 3490 device types. The device signals end-of-tape just before this limit is reached. This limit exists for both reading and writing tapes.
- Device types 3420 and 3590 do not have specific limits.

Emulated tape volumes created through this device manager are in *awstape* format and may be exchanged with other systems that can process this format. Note that all awstape files are compatible with all zPDT emulated tape devices. An awstape file written by an emulated 3590 can be read by an emulated 3420, for example.

The proper responses for hardware compaction (IDRC) are emulated, although tape data is not actually compacted by this method. The awstape data may be optionally compacted by the awstape device manager. This is controlled through a devmap or an awsmount parameter. The compaction format is unique to zPDT awstape. The default uncompacted form should be used for data interchange with other systems that use awstape data.

The awstape volumes are created when they are written to; that is, it is not necessary to create or initialize the volume before writing to it.

4.3.5 The awsosa device manager

The awsosa device manager emulates various OSA-Express[17] functions, as used by System z TCP/IP or SNA.[18] Two manager formats are used:

```
[manager]
name awsosa 8888 --path=F0 --pathtype=OSD [--interface=xxxx]
device 400 osa osa --unitadd=0
device 401 osa osa --unitadd=1
device 402 osa osa --unitadd=2



[manager]
name awsosa 2345 --path=A0 --pathtype=OSD [--interface] --tunnel_intf=y
[--tunnel_ip=10.1.1.1] [--tunnel_mask=255.0.0.0]
device 404 osa osa --unitadd=0
device 405 osa osa --unitadd=1
device 406 osa osa --unitadd=2
```

The first example would be used with a typical PC Ethernet adapter. The second format is used for a tunnel interface between the emulated OSA adapter and the underlying Linux TCP/IP system. The awsosa device manager can concurrently use the same Ethernet adapter that is used by Linux for normal Linux TCP/IP functions, but the OSA user and Linux cannot communicate through it. That is, both OSA and Linux can share the adapter for connection to external TCP/IP systems, but they cannot communicate with each other. A tunnel interface (which is similar to another Ethernet adapter) is needed for direct communication between the underlying Linux system and the System z OSA operation.

The `--path` operand specifies a CHPID number. The correct number is determined with the **`find_io`** `command`. For these examples we assume the CHPID for Ethernet is F0 and the CHPID for a tunnel interface is A0. The `--pathtype` is OSD (for QDIO) or OSE (for LCS or

[17] Larger systems have OSA-Express, OSA-Express2, and OSA-Express3 channels. The awsosa device manager provides a subset of these channel functions.
[18] SNA usage has not been tested by IBM and is not supported for zPDT usage.

non-QDIO). In some cases the **find_io** command does not provide a CHPID (path name) for a LAN interface. The `--interface=xxxx` parameter may be used to name a specific LAN interface. The interaction of the `--path` and `--interface` parameters is explained in detail in Chapter 10 in the most recent editions of the third book in this series (SG24-7723). An example of using the `--interface` parameter might be:

```
name awsosa AAAA --path=B0 --pathtype=OSD --interface=em1
```

The `--unitadd` operands specify the internal OSA interface number; normally these are not needed for QDIO operation. They may be needed for non-QDIO operation if more than one TCP/IP interface is used. z/OS TCP/IP requires three OSA addresses for QDIO operation.

SNA usage would require CHPID type OSE, although SNA usage with zPDT is not supported. The z/OS device type should be OSA, as seen in the z/OS IODF (and when displaying devices on the MVS console).[19] When used in OSE mode, the OSA interfaces are associated with OAT[20] definitions that specify how each interface is to be used.

Examples of OSA setup are in Volumes 2 and 3 of this documentation series. The limits in Table 4-1 apply to OSA-Express emulation.

Table 4-1 OSA-Express limits, per port

Maximum OSAs (and maximum OSA CHPIDs)	4
Maximum home addresses (IPv4 + IPv6 + DVIPA) per OSA port	64
Maximum IPV6 addresses	32
Maximum multicast addresses (IPv4 + IPv6)	64
ARP table size	256
IP stacks per port (OSD or OSE)	16
SNA PUs per OSA-Express port (SNA is not supported for zPDT)	512
OSE subchannels per stack	2
OSE or OSD maximum devices	48
OSE IP stacks per OSA port/CHPID	16
OSD subchannels per stack	3
OSD subchannels per OSA/CHPID	48

4.3.6 The awsrdr device manager

The awsrdr device manager emulates a 2540 card reader. Only one awsrdr device may be configured for an instance of zPDT operation. Typically, the emulated card reader is used to submit jobs to the operating system.[21] If we assume this to be z/OS, then JES2 or JES3 should be configured with a “hot” reader.[22] The traditional address for a 2540 is 00C, and we use this in our examples.

[19] Older z/OS systems may use CTC device definitions for these interfaces, especially when they are used for TCP/IP. These definitions should be replaced with device type OSA.

[20] An OAT is an OSA Address Table.

[21] In principle, we could directly allocate the card reader to a job using the appropriate DD statement. We did not try this.

[22] The term “hot reader” means there is always a read outstanding for the card reader. As soon as an operator places cards in the reader, JES begins reading them.

The awsrdr device manager monitors the directory specified in the devmap. When a file is found in the directory it is read (assuming a System z program has a *read* outstanding for the card reader, as would be the case with a JES hot reader). After the file ("card deck") is read it is moved to the *old* subdirectory. In this way there is never a file in the directory assigned to the reader, other than a file someone has just moved there to be read. As soon as it is read, it is moved out of the reader directory. If awsrdr is not active, or if there is no System z program trying to read cards, then files sit in the reader directory indefinitely.

The devmap entry for the card reader could appear like this:

```
[manager]
name awsrdr 010C
device 00C 2540 2821 /home/ibmsys1/cards/*
```

When a file is moved into the /home/ibmsys1/cards/ directory (using a Linux utility to move the file) it is then ready to be read by the emulated card reader. After the file is read by the card reader, it is moved to the /home/ibmsys1/cards/old/ directory. The /home/ibmsys1/cards/ directory we mention is just an example, of course. We can specify any path name (but the path must exist). The default path is /home/<userid>/z1090/cards/ and this is used if no path is specified in the devmap.

ASCII and EBCDIC

Linux text files are normally in ASCII. z/OS cards are normally in EBCDIC, but may contain binary information. A card reader uses fixed-length records (80 bytes) but a Linux text file has variable length records terminated with an NL character.

The conversion rules are as follows:

- If the input file name (in the directory used by `awsrdr`) contains the suffix `.ebc` or `.bin`, then the file is assumed to already be in EBCDIC and no translation is done.
- If the input file contains the suffix `.txt` or `.asc`, then the file is assumed to be in ASCII and is converted to EBCDIC.
- If the input file contains the ASCII characters // or ID or $$ or USERID in the first bytes, the file is assumed to be in ASCII and is converted to EBCDIC.
- If none of these conditions are true (suffix .ebc, or .bin, or .asc, or .txt, or recognizable first characters in ASCII), then the file is assumed to be EBCDIC (or binary as used for System z) and is not converted.
- If a file is converted from ASCII, the record length is padded with blanks to 80 bytes and the terminating NL bytes are removed.
- If the file is not converted from ASCII, for one of the reasons listed here, then awsrdr reads it in 80-byte chunks and passes the data (unchanged) to the emulated card reader.

Another way to translate ASCII text files to EBCDIC card files is with the **`txt2card`** command.

The ASCII/EBCDIC translation table is fixed in all cases.

4.3.7 The awsprt device manager

The awsprt device manager emulates a 1403 or 3211 printer. FCB functions are supported (for 3211 emulation), but UCS functions (for a 1403) are not supported. A fixed translation table is used to convert EBCDIC to ASCII. The device manager automatically inserts NL characters between output records. Unprintable characters are translated to blanks and no *unit check* is generated for these.

awsprt cannot recognize divisions between System z jobs. It simply concatenates all output (potentially from multiple jobs) into the output file. The devmap specifies the output file to be used:

```
[manager]
name awsprt 0003  [--windows]
device 00E 1403 2821 /home/ibmsys1/print
```

If a file name is not provided with the device statement, then the default file name (/home/<userid>/z1090/listings/dev-nnnn.lst) is used. The **--windows** option causes the output lines to be terminated with CR/LF characters instead of NL characters.

The **awsmount** command may be used to close the existing output file and open a new output file. The previous output file is closed properly, and is then available for display or printing under Linux.

4.3.8 The awscmd device manager

This device manager provides a "device" that appears to System z software as a tape drive. Its function is to send commands (and data) to the underlying Linux and then receive the output from the Linux command. Any Linux command may be sent, including those that could destroy the Linux system.[23] Obviously, this device manager should be used with care and may not be appropriate for a zPDT environment that can be accessed by untrusted users.

Configuration is similar to other device managers:

```
[manager]
name awscmd 20
device 580 3480 3480
```

The device type can be 3420, 3422, 3480, 3490, or 3590; these are the tape device types emulated by zPDT. The device number (580) should match a corresponding device type in your z/OS IODF. (Any device number may be used with z/VM.)

The intended operation (by a System z application program) is as follows:

1. A rewind is issued to the device.
2. The desired Linux command (expressed in EBCDIC) is written to the device.
3. Any stdin data to be used by the Linux command is written to the device.
4. EBCDIC to ASCII translation is done automatically, with a fixed translation table.
5. A tape mark is written to the device.
6. At this point, the awscmd device manager submits the command (and data) to Linux through a shell that does not appear on the Linux screen. The current Linux directory for the command is the directory that was used to start zPDT.
7. When the awscmd function completes there are four files on the pseudo-tape device:
   - The command file that was submitted to Linux (with redirection operands that were automatically added by awscmd)
   - The stdout data from the Linux command
   - The stderr data from the Linux command
   - The return code (converted to characters) from the Linux command
8. The output (on the pseudo-tape) has been converted to EBCDIC.
9. Two tape marks are at the end of the pseudo-tape.

[23] The Linux commands are executed with the authority of the userid that started zPDT operation.

Restrictions

The command you send to Linux cannot include any redirection (< or > characters), asynchronous indicator (& character), or pipe ("|" or vertical bar character). The pseudo-tape device will appear to be busy while Linux is executing the command. Any Linux command that creates substantial delays (of many seconds) may cause I/O timeout errors to be generated in z/OS.

At the time of writing, some characters did not survive the EBCDIC to ASCII conversion when included in SYSIN data. These were the tilde (~), caret (^), colon (:), double quote ("), less-than (<), greater-than (>), and question mark (?). This restriction may change in later versions of awscmd.

An extended example of awscmd usage is in the third document in this series (SG24-7723).

4.3.9 The awsscsi device manager

The awsscsi device manager emulates a mainframe tape drive using a SCSI tape drive. The only tested and supported drives are Fujitsu M2488E units (compatible with IBM 3490 and 3490E cartridges), IBM LTO3 and LTO units, IBM TS1120[24], and IBM 3592 (Fibre Channel interface) units.

```
[manager]
name awsscsi 700
device 581 3490 3490 /dev/sg5
```

The last operand of the device statement denotes the SCSI device to be used. This must be given as a /dev/sgx name, and not as a /dev/stx name. The differences are complex; Volume 3 describes methods for determining the correct /dev/sgx name. The SCSI tape drive appears as an IBM 3490 to the System z software.[25] Note that proper IBM 3490 characteristics, such as a maximum block count, are emulated and may produce unexpected results.

The **awsmount** command may be used with SCSI tape devices.

4.3.10 The aws3215 device manager

The aws3215 device manager provides emulation of a 3215 console.

```
[manager]
name aws3215 AC00
device 009 3215 3215
```

It is possible, but very unusual, to have multiple 3215 devices. Input to the 3215 console is via the **awsin** command, entered in a Linux command window. Output appears in the Linux window used for the **awsstart** command.

4.3.11 The awsoma device manager

The awsoma device manager is used to read CDs or DVDs[26] written in a special format known as OMA. This is for input only; it is not possible to write to an awsoma device. In earlier

[24] The IBM TS1120 might not handle blocks larger than 32K. This is a restriction of some adapter cards, and not of the TS1120 drive itself. The user must determine if these limitations exist for his adapter.

[25] Remember that IBM 3490 tape units have their own characteristics. One of these is a maximum block count of approximately 4 million (a 22-bit number).

[26] It is possible to have OMA files on other media, but a CD or DVD is usually where they are found.

days, VM and VSE were available in OMA format; some Linux distributions for System z may use this format.

```
[manager]
name awsoma D000
device F00 oma oma /media/ROM/;xyz.tdf
```

The variable portion of the device statement (after the second *oma*) must be in a specific format, with two names separated by a comma or semicolon. There must be no blanks between the operands. The first name is a path name and the second name is a particular file name. That is, the second name is *relative* to the path specified by the first name.[27]

In a Linux-based zPDT system, the net effect is that the two names are concatenated; in the example above, the effective file name used for input to awsoma would be `/media/ROM/xyz.tdf`. The slash (/) after ROM could be omitted and a slash inserted before `xyz.tdf`; this would result in the same effective file name.

Releases of zPDT code later than 39.14 have expanded the possible formats to include the following:

```
device 123 oma oma  /tmp/;my.tdf      results in /tmp/my.tdf
device 123 oma oma  /tmp/my.tdf       single fully qualified name
device 123 oma oma  my.tdf            results in /home/ibmsys1/my.tdf
      (assuming zPDT was started from /home/ibmsys1)
device 123 oma oma  /media/myCD/TAPES/my.tdf
      (data is assumed to be in /media/myCD/xxxxx)
```

The first example here follows the original requirements. The second example uses a single fully-qualified name. The third example causes the specified file name (my.tdf) to be relative to the directory used to start zPDT operation. The last example depends on the keyword TAPES to indicate that data files are relative to the directory above TAPES.[28]

The variable portion of the device statement may be omitted. In this case, **awsmount** commands are used to associate the TDF file with the awsoma device. The two-operand format, as used in the initial description above, is not valid for awsmount.

4.3.12 The awsctc device manager

The awsctc device manager emulates a 3088 channel-to-channel control unit. A typical definition is as follows:

```
[manager]
name awsckd 5432
device E40 3088 3088 ctc://192.168.0.81:3088/E42
                                 |        |   |
                                 |        |   + remote device number
                                 |        + remote port number
                                 + remote IP address
```

Multiple devices may be defined for this device manager. A separate chapter in the third book in this series (SG24-7723) describes the setup and usage of this device manager.

[27] This operand convention was evolved for early OS2-based machines, where it helped deal with drive letters that might be needed before a file name.

[28] This convention was used in the original OMA support and is documented in IBM publication SC53-1200.

4.4 zPDT commands

zPDT commands are entered as normal Linux line commands in a Linux terminal window. If zPDT is running (that is, the System z function is operating) then zPDT commands directed to the System z must be entered in a Linux window that is owned by the Linux userid that started the System z function.[29] This is not normally an issue unless multiple zPDT instances are running, each under a separate Linux userid.

The term *devmap* is used throughout these documents to indicate a zPDT device map, which is a simple Linux text file that specifies System z characteristics and emulated I/O devices for an instance of System z operation.

The return values listed for many of the commands are normally not relevant, but might be used if the commands are embedded in a shell script, for example.

Note that most of the commands have a *help* option, usually invoked by an **-h** operand.[30] This operand is not shown in the following descriptions because it is the same for all commands and would add needless bulk to the command descriptions. The same help information may be obtained with a Linux **man** command using the zPDT command name as the operand. For example:

```
$ man awsstart                (request MAN pages for awsstart command)
$ awsstart -h                 (displays the same MAN pages)
```

The $ in this example (and in many examples throughout these documents) represents the Linux prompt.

4.4.1 adstop

The **adstop** command sets an address stop point for the default processor. When the instruction address in the PSW equals the specified address, the CP enters a stopped state. The PSW check is effective for both virtual or real addresses. Only one stop address may be in effect for each CP. To be most effective, only one CP should be in use or the same address stop should be set for all active CPs. (The default CP is changed with the **cpu** command.) zPDT must be operational when using this command.

```
adstop hex-address [on | off]
                   [q]
```

Where:

`q` - query the current settings for the command.

The return values are:

```
0     The address stop was set.
16    Unable to initialize the manual operations interface.
69    Unusable hex address.
101   The address stop was not set.
```

Command examples are:

```
$ adstop 4FCC
$ adstop off
```

29 This means the same Linux user who issued the **awsstart** command must enter any additional zPDT commands that affect that instance of System z operation.

30 In some cases, a ? operand (question mark) can be used in addition to the -h operand.

4.4.2 The alcckd command

The **alcckd** command creates (and formats) a Linux file that may be used as an emulated 3380 or 3390 DASD unit. The file is formatted to correspond to 3380 or 3390 tracks and cylinders in CKD format, but is otherwise not initialized. A utility program (such as ICKDSF) must later be used to create a volume label, VTOC, and so forth. A standard model (3380-1, 3380-2, 3380-3 or 3390-1, 3390-2, 3390-3, 3390-9) may be specified to establish the size of the emulated device, or a specific number of cylinders may be specified to create a nonstandard size. zPDT need not be operational when using this command. (zPDT would need to be restarted, with an updated devmap, to use the newly created CKD device.)

```
alcckd file-name { -ddevice-type [-snumber-of-cylinders] [-q] }
                 { -r                                         }
                 { -rs                                        }
                 { -rf                                        }
                 { -ve | -vr | -vc | -vi | -vd }
```

Where:

`file-name` is a Linux file name.

`-ddevice-type` is a device type, optionally with a model number.

```
-d3380 - device type 3380 (size specified by the -s parameter)
-d3380-1 - device type 3380 with 885 cylinders
-d3380-2 - device type 3380 with 1770 cylinders
-d3380-3 - device type 3380 with 2655 cylinders
-d3390 - device type 3390 (size specified by the -s parameter)
-d3390-1 - device type 3390 with 1113 cylinders
-d3390-2 - device type 3390 with 2226 cylinders
-d3390-3 - device type 3390 with 3339 cylinders
-d3390-9 - device type 3390 with 10017 cylinders
```

`-snumber-of-cylinders` (used when a standard model is not specified) determines the number of cylinders to be created. The maximum size is 65520 cylinders for a “normal” 3390, or 268,435,456 cylinders for a “large” 3390.

`-r` displays the CKD device attributes for an existing emulated CKD file. The **alcckd** command with no operands produces the same information.

`-rs` displays the CKD device attributes for an existing emulated CKD file and scans the file to verify that the emulated CKD formatting is correct.

`-rf` performs the -rs function and reinitializes any emulated tracks with incorrect formats; the contents of that track are lost.

`-q` invokes quiet mode, with no output messages to the Linux terminal.

`-ve`, `-vr`, `-vc`, `-vi`, and `-vd` are related to versioning and are described in Volume 3 of this documentation series.

Earlier releases of zPDT did not allow a space between the -d or -s flag and the associated parameter. This restriction no longer exists, but examples are still in the *no space* format.

Earlier releases of this command had a -z option that caused all emulated tracks to be written; the default action was to write a Linux *sparse* file. This is now changed and **alcckd** always writes the full tracks for the emulated volume.

If more than 65520 cylinders (for a 3390) are specified, an extended address volume (EAV) is produced. The number of cylinders in an EAV should be an even multiple of 1113.

The return values are:

0	Successful operation.
11	Insufficient Linux disk space to create the file.
12	Linux path not found.
13	Linux write protection (permissions) error.
14	General error.
15	Specified file already exists.
16	File not found or file name is invalid.
17	Drive not ready.
19	Disk not valid.
20	Not an emulated CKD volume.
21	Emulated CKD format is not valid

Examples of command usage:

```
$ alcckd /z/WORK01 -d3390-3          (create new emulated 3390-3 volume)
$ alcckd /tmp/222222 -d3390 -s100    (create small 3390 volume, 100 cylinders)
$ alcckd /z/WORK01 -rs               (verify format of CKD volume)
```

4.4.3 The alcfba command

The **alcfba** command creates (and formats) a Linux file that may be used as an emulated 9336 DASD unit. The file is formatted to correspond to the fixed blocks of a 9336 device and a volume name may be assigned. A standard model (9336-1, 9336-2) may be specified to establish the size of the emulated device, or a specific number of blocks may be specified to create a nonstandard size. (Fixed-block devices compatible with 9336 drives may also used these emulated volumes.) zPDT need not be operational when using this command. (zPDT would need to be restarted, with an updated devmap, to use the newly created FBA device.)

```
alcfba file-name {-ddevice-type [-ssize{B|K|M}][-vvolser][-q] }
                 {-c -vvolser                            [-q] }
                 {-r                                          }
```

Where:

`file-name` is the Linux file name for the emulated volume.

```
-ddevice-type:
   -d9336 - device type, size is set by the -s parameter.
   -d9336-1 - device type, size is 920,115 blocks.
   -d9336-2 - device type, size is 1,672,881 blocks.
```

`-ssize` is the size (in decimal) of the emulated volume.

```
   -snnnB specifies the number of 512K blocks for the device.
   -snnnK specifies the total volume size in kilobytes.
   -snnnM specifies the total volume size in megabytes.
```

`-vvolser` sets the volume serial to the indicated name (6 characters). The volser is six characters and automatically converted to upper case.

`-c` change the volser of an existing FBA volume.

`-q` sets quiet mode with no output messages sent to the Linux terminal.

`-r` display the attributes of an existing FBA volume.

Earlier releases of zPDT did not allow a space between the -d, -s, or -v flag and the associated parameter. This restriction no longer exists, but examples are still in the *no space* format.

Return values are:

```
0     Command completed successfully.
1     Help information was displayed.
11    Insufficient Linux disk space to create the FBA volume.
12    Path not found.
13    Write protection (permissions) error.
14    General error.
15    Specified file already exists.
16    File not found or the file name is not valid.
17    Drive not ready.
19, 20   Disk not valid.
```

Command examples are:

```
$ alcfba /z/TEMP01 -d9336-1 -vSCRTCH
$ alcfba /tmp/444444 -d9336 -s2000B -vMYVOL1
$ alcckd /z/TEMP01 -c -vWORK99
```

4.4.4 The ap_create command

The **ap_create** command dynamically creates an emulated cryptographic processor. zPDT must have been started when this command is used.

```
ap_create -a n
```

Where:

`n` is the number of the coprocessor and is in the range 0 - 15.

Emulated cryptographic coprocessors are normally specified in the devmap, in the [adjunct-processors] stanza and are created automatically when zPDT is started. This command would be used only in unusual situations.

The ap_destroy command

The **ap_destroy** command removes an emulated cryptographic coprocessor if it is not connected to a CP process. zPDT must have been started when this command is used.

```
ap_destroy -a n
```

Where:

`n` is the number of a defined cryptographic coprocessor.

Emulated cryptographic coprocessors are automatically removed when zPDT is stopped. This command would be used only in unusual circumstances.

4.4.5 The ap_query command

The **ap_query** command displays the status of emulated cryptographic coprocessors. zPDT must have been started when this command is used.

```
ap_query
ap_query -a n
```

Where:

`n` is the number of a defined cryptographic coprocessor.

This command queries basic status and domain information. With no operand, it lists the coprocessors available to System z. With an operand, it lists which domains are used by the indicated coprocessor.

4.4.6 The ap_von and ap_voff commands

The **ap_von** and **ap_voff** commands vary emulated cryptographic coprocessors (or domains) online or offline. zPDT must have been started when this command is used.

```
ap_von -a n
ap_von -a n -d y
ap_voff -a n
ap_voff -a n -d y
```

Where:

`n` is the number of a cryptographic coprocessor.

`y` is the number of a domain within the specified coprocessor.

Emulated cryptographic coprocessors defined in the devmap are automatically made online when zPDT is started. The **ap_von** and **ap_voff** commands are not normally used, although they become relevant when **ap_create** or **ap_destroy** commands are used.

4.4.7 The ap_vpd command

The **ap_vpd** command displays Vital Product Data (VPD) data for an emulated cryptographic coprocessor. zPDT must have been started when this command is used.

```
ap_vpd -a n
```

Where:

`n` is the number of a defined cryptographic coprocessor.

This command might be useful to verify that the specified coprocessor is, indeed, active. The data displayed is not relevant to normal zPDT operation.

4.4.8 The ap_zeroize command

The **ap_zeroize** command erases (zeros) the content of a specified emulated cryptographic coprocessor, or a subset of a coprocessor. zPDT must have been started when this command is used.

```
ap_zeroize -a n -d y
ap_zeroize -a n -i
```

Where:

`n` is the number (0-15) of an emulated cryptographic coprocessor.

`y` is a domain (0-15) in the specified coprocessor.

This command reinitializes (zeros) all the data, such as keys, that is retained by the coprocessor. The first version of the command (with the **-d** operand) affects only the specified domain in the specified coprocessor. The second version (with the **-i** operand) zeros the whole adapter. Either **-i** or **-d** must be specified (with an appropriate domain number for y).

When a new cryptographic coprocessor is used (or when one is zeroized) it must be reinitialized. This is normally done with the ICSF utility, as explained in the third book in this documentation series.

4.4.9 The attn command

The **attn** command creates a simulated unsolicited device end interrupt from a device.

```
attn device-number
```

Where:

`device-number` is the address (device number) of a device in the current devmap.

An unsolicited device end is also known as an asynchronous attention interrupt. The meaning of an attention interrupt varies depending on the device type. In typical zPDT operation this command is probably not used.

A command example is:

```
$ attn 590
```

4.4.10 The aws_bashrc and aws_sysctl commands

These commands may be used during zPDT installation to bypass making tedious manual changes to Linux files. These two command scripts are located in `/usr/z1090/bin`, along with all the other zPDT command files. However, at the time these two commands are typically used, `/usr/z1090/bin` is not yet in the Linux PATH. For that reason, these commands are typically called by their full path name:

```
# /usr/z1090/bin/aws_sysctl      (change to root before using this command)
$ /usr/z1090/bin/aws_bashrc      (do not use this command as root)
```

The **aws_sysctl** command makes required modifications to `/etc/sysctl.conf` and then executes **/sbin/sysctl**. The **aws_sysctl** command must be executed with *root* authority. The statements this command adds to `/etc/sysctl.conf` are appropriate for many zPDT users, but may need to be manually modified for especially large zPDT instances. This is discussed further in the second document in this series (SG24-7722).

The **aws_bashrc** command modifies the `.bashrc` file in the current directory. You should normally be in your home directory (and *not* as root) when executing this command. The command adds the appropriate zPDT PATH statements to `.bashrc`.

4.4.11 The awsckmap command

The **awsckmap** command validates the content and format of a device map, reporting any errors found. zPDT need not be operational when using this command.

```
awsckmap devmap-name [--list]
                     [--sys ]
                     [--sum ]
                     [--mgr ]
                     [--dev ]
```

Where:

`devmap-name` is a Linux file name (fully qualified, if necessary).

`--list` causes the command to output a listing of the complete configuration.

`--sys` provides information about the systems section of the devmap.

`--sum` provides information about the subchannel/devices in the devmap.

`--mgr` lists the device managers required by this devmap.

`--dev` lists detailed device information from the devmap.

The return code is always zero. Examples of the command are:

```
$ awsckmap aprof1
$ awsckmap /z2/VM/devmap2.txt --list
```

4.4.12 The awsin command

The **awsin** command provides input to an emulated 3215 console. The address (device number) of the 3215 must be provided if more than one 3215 is defined. (Note that 3215 device usage is rare today, and this command is seldom used.) zPDT must be operational when using this command.

```
awsin { [dev-address] 'text' }
      { [dev-address] -a     }
```

Where:

`dev-address` is the address (device number) from the devmap.

`'text'` is the message to be sent to the 3215.

`-a` indicates that an attention interrupt should be sent, but no text.

The text operand is normally included in single quotes to prevent the Linux shell from altering it. Return values are:

```
0     Input text queued for input or attention interrupt sent.
-1    Errors. (Devmap problem; -a and text both included; text too long)
-2    No 3215 device found in the devmap.
-3    No dev-address specified and multiple 3215s exist in devmap.
```

A typical example of command usage is:

```
$ awsin 'sta,id=ifdasd'
```

4.4.13 The awsmount command

The **awsmount** command associates a Linux file with an emulated I/O device. It can also be used to perform various operations on emulated tapes, query device status, and make a device read-only or read-write. zPDT must be operational when using this command.

```
awsmount dev-address {-b | --bsf [n]                                 }
                     {-c | --compress                                }
                     {-f | --fsf [n]                                 }
                     {-s | --rew                                     }
                     {-t | --wtm [n]                                 }
                     {-x | --run                                     }
                     {-u | --unmount                                 }
                     {-r | --ro | --readonly                         }
                     {-w | --rw | --readwrite                        }
                     {-q | --query                                   }
                     {{-o | --replace} file-name [-r|--ro|-w|--rw]          }
```

```
                         {{-m | --mount  } file-name [-r|--ro|-w|--rw]         }
                         {-d | --disc | --disconnect                           }
```

Where:

`dev-address` is the device address from the devmap.

`-b` or `--bsf` backspaces over one tape mark on an emulate tape drive.

`-c` or `--compress` causes output to an emulated tape drive to be compressed.

-f or --fsf forward spaces over one tape mark on an emulated tape drive.

`-s` or `--rew` rewinds an emulated tape drive.

-t or `--wtm` writes a tape mark on an emulated tape drive.

`-x` or `--run` produces a rewind and unload on an emulated tape drive.

`-u` or `--unmount` produces an unmount operation on the device. This removes any previous Linux file association with the device.

`-r` or `--ro` or `--readonly` makes the emulated device read-only.

`-w` or `--rw` or `--readwrite` makes the emulated device read-write.

`-o` or `--replace` replaces the existing file association with a new file association (similar to replacing a tape on a tape drive) and the new file has the indicated read-only or read-write characteristics.

`-m` or `--mount` associates a new file with the emulated device, when no file was associated with it at the time of the command.

`n` is the number of operations to perform. (This option is not available yet.)

`-d` or `--disc` or `--disconnect` is used to force disconnection of a 3270 session.

Tape operations (bsf, fsf, rew, wtm, and run) for emulated tape drives also may be used with SCSI-attached tape drives. Appropriate **awsmount** functions may be used for the awsckd, awsfba, awstape, awsscsi, awsprt, and awsoma device managers. The **awsmount** command should never be directed at an awsosa device.

Examples of extended use of awsmount (using 580 as a typical device number) are:

```
For tape drives (emulated or SCSI)
   awsmount 580 -q                          query currently mounted file
   awsmount 580 -m  /tmp/tapevol/123456     mount emulated volume
   awsmount 580 -o  /z/654321               replace mounted volume
   awsmount 580 -u                          unmount current volume
   awsmount 580 -x     (or --run)           unmount current volume
   awsmount 580 -b                          backspace over tape mark
   awsmount 580 -f                          forward space over tape mark
   awsmount 580 -s                          rewind tape volume
   awsmount 580 -t                          write tape mark
   awsmount 580 -c /tmp/mytape1             mount and use compression
For OMA tapes (using device number 180 as an example)
   awsmount 180 -q                          query currently mounted file
   awsmount 180 -m /tmp/oma/11111           mount emulated volume
   awsmount 180 -o /z/oma/dosvol            replace mounted volume
   awsmount 180 -u                          unmount current volume
   awsmount 180 -x    (or --run)            unmount current volume
   awsmount 180 -b                          backspace over tape mark
   awsmount 180 -f                          forward space over tape mark
   awsmount 180 -s                          rewind tape volume
```

```
Disks and printers (using 300 and 00E device numbers)
   awsmount 300 -q                            query mounted file name
   awsmount 300 -m  /z/LOCAL1                 mount emulated volume
   awsmount 00E -m  /tmp/print1               printer output file
   awsmount 300 -o  /z/LOCAL2                 replace mounted volume
   awsmount 300 -u                            unmount current volume
aws3270 (local 3270 sessions; device number 702 for example)
   awsmount 702 -q                            query tn3270 client
   awsmount 702 -d                            force a disconnect
awsscsi (connect SCSI tape drives, using device number 580 for example)
   awsmount 580 -m /dev/sg3                   connect SCSI tape drive
```

4.4.14 The awsstart command

The **awsstart** command starts zPDT operation by creating a System z environment.

```
awsstart [--noosa][--map] file-name [--clean] [--localtoken]
```

Where:

--noosa creates a zPDT environment without any OSA components.

--map is optional before the file name.

`file-name` is the name of a device map file.

--clean causes all previous logs and traces to be deleted.

--localtoken causes a local USB token to be used while retaining a previous serial number.

Use of the **--noosa** parameter would be unusual and should be done only at IBM direction. zPDT maintains a variety of logs and traces in the ~/z1090/logs directory. Note that this is a subdirectory of the userid that installed and now starts zPDT. The contents of the logs directory can grow over time. If no zPDT problems are under investigation, using the --clean parameter will ensure that only currently relevant logs and traces (from the zPDT instance just being started) will appear in the directory.

A zPDT system may be configured to use a remote license manager. If the owner wants to temporarily use a local token without changing the remote license manager configuration details, the --localtoken option may be used. The details may be found in the third book in this series (SG24-7723).

The only defined return value is zero. An example of the command is:

```
$ awsstart devmap3 --clean
```

4.4.15 The awsstat command

The **awsstat** command queries the status of emulated I/O devices.

```
awsstat [-i [n-seconds]] [device-list]
```

Where:

`-i n-seconds` indicates the list should be repeated every n-seconds. If the n-seconds parameter is not provided, the default is 400 seconds.

`device-list` is list of device numbers. If no device-list is provided, all defined emulated devices are listed. A range of device numbers may be specified, or the name of a device manager.

The device-list may use three- or four-digit hexadecimal operands. These are the device numbers (“addresses”) defined in the current devmap. The output display for emulated disk devices includes the current head position (cylinder, track) on the device.

If the interval option (-i) is used, there is a help panel (accessed by entering `h` or `?`) that allows the output to be sorted. Entering **q** during an interval will terminate the command.

The defined return values are:

```
0     Command complete.
-2    Unable to locate or open devmap.
-3    Unable to access shared device status memory.
-4    Insufficient memory to initialize the command.
-5    Unable to collect device status.
```

An example of the command and resulting output is:

```
$ awsstat 700,a80
Config file: /home/ibmsys1/aprof9  IOCDS:none, 3270port:3270
DvNbr S/Ch --Mgr--- Actv Busy --PID-- ------Device information-------------
0700     0 AWS3274   Yes   No    4315 IP-127.0.0.1      Term-mstcon, Avail-No
0a80     5 AWSCKD    Yes   No    4449 Cyl-2036, Head-3  /z/Z9RES1
```

The S/Ch column lists the subchannel number (internal to zPDT). Each device is represented by a Linux process and the process IDs are listed. The IOCDS note in the header should be ignored.

```
$ awsstat a80-a85          (a range of device numbers)
$ awsstat awsckd           (all devices owned by this device manager)
```

4.4.16 The awsstop command

The **awsstop** command ends zPDT operation. This operation ends abruptly, with no warning to the System z operating system.

```
awsstop
```

There is no return value. An example of the command is:

```
$ awsstop
```

4.4.17 The card2tape command

The **card2tape** command copies a Linux text file to an emulated tape volume, in card image format. zPDT need not be running to use this command.

```
card2tape [-c        ]  [-a     ]  inputfile outputfile
          [--compress]  [--ascii]
```

Where:

`-c` or `--compress` causes the output awstape file to be compressed.

`-a` or `--ascii` indicates the input file is ascii and causes the output to be translated to EBCDIC.

The compression option saves space in the emulated output file, but is not compatible with other platforms that may use awstape files. It does not indicate the use of hardware tape compression, such as IDRC. The output is in 80-byte records, blanks appended to input records if necessary.

The default conditions for ASCII to EBCDIC translation are the same as used for the awsrdr device manager, and are described in 4.3.6, “The awsrdr device manager” on page 44. The -a or --ascii parameters must be used to force translation. The EBCDIC/ASCII translation table used cannot be changed.

No return values are defined. An example of the command is:

```
$ card2tape --ascii myfile.txt myfile.awstape
```

4.4.18 The card2txt command

The **card2txt** command creates an ASCII text file from an EBCDIC input file in card format. zPDT need not be running when this command is used.

```
card2txt input-file output-file
```

Where:

`input-file` is an EBCDIC file that must be an exact multiple of 80 bytes long.

`output-file` is the name of the Linux text file.

The input file is read in 80-byte blocks and each block is assumed to be a card record. Trailing blanks are then removed from each 80-byte block and a NL (NewLine) character added, as used for a Linux text file. The EBCDIC/ASCII translation table used cannot be changed.

No return values are defined. An example of the command is:

```
$ card2txt carddeck.ebc file23.txt
```

4.4.19 The ckdPrint command

The **ckdPrint** command dumps (prints) the contents of an emulated disk drive (such as a 3390) to Linux stdout. zPDT need not be running when this command is used.

```
ckdPrint emulation-file-name
```

Where:

`emulation-file-name` is the name of the Linux file that contains the emulated disk.

The program prompts for the range of tracks to dump. These are entered as four decimal numbers separated by blanks. The numbers are:

- The starting cylinder number
- The starting head number
- The ending cylinder number
- The ending head number

After dumping the specified tracks, the prompt is repeated. Entering a null line ends the program. Cntl-C may be used to terminate the program. Count, key, and data fields are shown for each block on the track(s) that are dumped.

No return values are defined for this command. An example that dumps the contents of the first two tracks (track 0 and track 1) of the first cylinder (cylinder 0) is:

```
$ ckdPrint /z/Z9DIS1
DeviceType-3390, Cylinders-3339, Tracks/Cyl-15, TrkSize-56832
Input extent in decimal -- CC-low HH-low CC-high HH-high
0 0 0 1
```

4.4.20 The clientconfig command

The **clientconfig** command provides a menu function to assist in configuring a remote license server and Unique Identity Manager (UIM) that provides consistent System z serial numbers. This command must be run as *root*.

```
clientconfig [directory]
```

This command always operates on a file named `sntlconfig.xml`. By default, this file is in `/usr/z1090/bin` but a different directory name may be specified as an operand.

A description of these functions is lengthy and is found in Volume 3 of this documentation series, in the chapter titled "License & serial number servers."

4.4.21 The clientconfig_authority command

The **clientconfig_authority** command adds a Linux userid or removes a Linux userid from a list of userids that may issue the **clientconfig** command. Normal usage of **clientconfig** requires the user to operate as *root*. The **clientconfig_authority** command allows the installation to avoid usage of *root* when changing license server configurations. The **clientconfig_authority** command must be run as *root*, but it is used only once for a given userid.

```
clientconfig_authority [-a | -d] userid

-a adds the indicated userid to the list of userids that are allowed to issue
   the clientconfig command.
-d removes the indicated userid from this list.
```

A command example is:

```
# clientconfig_authority -a ibmsys1
```

4.4.22 The cpu command

The **cpu** command selects the default CP that is the target for subsequent commands. zIIPs, zAAPs, and IFLs are considered CPs for this function. zPDT must be operational to use this command.

```
cpu cp-address
```

Where:

`cp-address` is the number of the CP that becomes the default target.

CPs are numbered starting with 0 and increasing by one for every CP (or zIIP or zAAP or IFL) that is defined in the *processors* statement of the devmap. The default target CP is CP number zero. Each CP has its own registers, active address space, and so forth. This command would be used in order to examine registers and memory in a particular CP.

The defined return codes are:

```
0     The default CP was changed.
12    The specified CP address is not valid.
16    Unable to initialize the manual operations interface.
```

An example of using the command is:

```
$ cpu 1          (select second CP, which is CP number 1)
$ stop          (place default CP in stopped state)
$ d psw         (display PSW of the default CP)
$ start         (start the default CP again)
```

4.4.23 The d command

The **d** (display) command displays CP information, including registers, memory, and architecture mode. This information is displayed from the default CP, as set by the **cpu** command. CP 0 is the initial default CP. zPDT must be operational to use this command.

```
d {r                                                   }
  {p | psw                                             }
  {pfx                                                 }
  {g | gn                                              }
  {y | yn                                              }
  {x | xn                                              }
  {z | zn                                              }
  {vphex-addr [t] [.hex-len | decimal-len]             }
  {vshex-addr [t] [.hex-len | decimal-len]             }
  {vhhex-addr [t] [.hex-len | decimal-len]             }
  {vahex-addr [t] [.hex-len | decimal-len]  access-reg }
  {hex-addr [t] [.hex-len | decimal-len]               }
```

Where:

`r` displays the current architecture mode.

`p` or `psw` displays the current PSW.

`pfx` displays the prefix register.

`g` or `g`*n* displays the contents of the general purpose registers. If a particular register is not specified (by the *n* parameter) then all are displayed.

`y` or `y`*n* displays floating point registers.

`x` or `x`*n* displays control registers.

`z` or `z`*n* displays access registers.

`hex-addr` is an address in memory.

`.hex-len` is the amount of memory to be displayed (in hexadecimal).

`decimal-len` is the amount of memory to be displayed (in decimal).

`vp` displays primary virtual memory.

`vs` displays secondary virtual memory.

`vh` displays the home address space virtual memory

`va` displays virtual memory via an access register, which must be specified

`access-reg` is the number of an access register

`t` (just after an address) indicates both hex and character displays are wanted.

A memory address not prefixed with vp, vs, vh, or va displays data at the real memory address. Memory is displayed on 32-byte boundaries. If the specified address is not on a 32-byte boundary, the next lowest 32-byte boundary is used. Each memory line displayed ends with the protect key for that memory. As a general statement, the CP should be in a stopped state before any of these display functions are used.

The `vp` prefix can be shortened to `v`. Note that a hexadecimal length is separated from the address with a period; a decimal length is separated with a blank.

A virtual address is meaningful only if an address space is active at the instant of the display. When z/OS is in a wait state there may be no active address space. As a general statement, these commands are not useful for application programming debugging unless there is a way to stop the CP while the application is actively being executed.

The **`d psw`** command is most useful for examining disabled-wait-state codes.

The return values are:

```
0     Command complete.
30    No arguments specified.
```

Examples of use are:

```
$ d psw              (display PSW)
$ d g2               (display contents of general purpose register 2)
$ d 461244 32        (display 32 bytes at real address x'461244')
$ d 461244.C0        (display x'c0' bytes at indicated address)
$ d v458332 100      (display 100 bytes at indicated virtual address)
```

4.4.24 The fbaPrint command

The **`fbaPrint`** command dumps (prints) the contents of one or more sectors on an FBA emulated disk drive. zPDT need not be active to use this command.

```
fbaPrint emulation-file-name
```

Where:

`emulation-file-name` is the name of the Linux file containing the FBA volume.

The command will prompt for the range of block numbers to be dumped. These are entered as two decimal numbers, separated by spaces. When the dump is complete, the prompt is issued again. A null input line will terminate the command.

No return values are provided. An example of the command is:

```
$ fbaPrint /z/VSE123
0 1
```

4.4.25 The find_io command

The **`find_io`** command is used to identify potential OSA ports.

```
find_io
        Interface Current          MAC               IPv4             IPv6
 Path    Name      State           Address           Address          Address
------  -----  ---------------- -----------------  --------------  --------------
  F0     eth0    UP, RUNNING     00:26:2d:f7:3b:4e  9.56.64.36     fe80::226:2dff:fef7:3b4e%eth0
  F1     eth1    UP, RUNNING     00:12:0e:4b:eb:0d  9.56.64.18     fe80::212:eff:fe4b:eb0d%eth1
  F8     wlan0   UP, NOT-RUNNING 00:23:15:17:1f:24  *                 *
  F9     pan0    DOWN            36:e6:26:4e:1c:b4  *                 *
  *      br0     UP, RUNNING      a2:ac:36:48:19:d2  *              fe80::a0ac:36ff:fe48:19d2%br0

  A0     tap0    DOWN             02:a0:a0:a0:a0:a0  *                *
  A1     tap1    DOWN             02:a1:a1:a1:a1:a1  *                *
  A2     tap2    DOWN             02:a2:a2:a2:a2:a2  *                *
```

A path (or CHPID) name is shown for most (but not necessarily all) LAN interfaces; these have names such as F0, F1, A0, and so forth. Interface names are shown, for example eth0, tap0, wlan0. A path name is normally specified for the awsosa device manager; in some cases the interface name may be needed if no path name is shown by the **find_io** command.. Note that all Linux LAN interfaces, whether enabled or not, are detected. This may cause default path assignments to differ from previous zPDT releases.

The other data shown (State, MAC address, IPV4, and IPv6 addresses) are informational only. The IP addresses apply only to Linux; z/OS (or another System z operating system) may also address the interfaces with completely different IP addresses. All LAN interfaces known to Linux are shown; some of these may not be relevant or tested for zPDT usage. The MAC addresses for tap devices are artificial.

The use of path and interface parameters is explained in detail in Chapter 10 in the most recent editions of the third book in this series (SG24-7723-04).

4.4.26 The hckd2ckd, hfba2fba, and htape2tape commands

These are three client commands used with the migration utilities.

hckd2ckd - Used with both z/OS and z/VM to migrate a CKD DASD volume.

hfba2fba - Used only with z/VM to migrate an FBA DASD volume.

htape2tape - Used only with z/VM to migrate a tape volume to a zPDT awstape volume.

The general syntax of the client commands (entered on the Linux client machine, using a normal Linux command window) is:

```
hxxx2xxx  host[:port] outfile [-n         ][-v       xxxxxx][-u     aaaa]
                              [--norestart][--volser xxxxxx][--unit aaaa]

                              [-e    eof-count] [-n]
                              [--eof eof-count]
```

Where:

`host` - is the TCP/IP name of the system with the matching server program. This may be a dotted-decimal address or a domain name that can be resolved by Linux TCP/IP.

`:port` - is a TCP/IP port number to be used by both the client and server program. It defaults to 3990.

`outfile` - is a file name (on the current Linux) system where the migrated volume is placed (in awsckd, awsfba, or awstape format).

`-n` (when used with htape2tape) indicates the output awstape file is not to be compressed. (Compression is the default.)

`-v` or `--volser` indicates the 3380/3390 volume (on the remote z/OS system) that is to be copied (migrated).

`-u` or `--unit` indicates the address (device number) of the volume that is to be copied (migrated).

`-e` or `--eof` indicates the number of consecutive tape marks that will indicate the end of the input tape. This is used only with z/VM tapes. The default is two tape marks.

Either the `-u` or `-v` parameter must be supplied for DASD, but not both; the `-u` parameter would normally be used for tapes. These commands are described in more detail in the chapter about the migration utility in the third book in this series (SG24-7723).

Examples of commands that could be used to run the client are:

```
$ hckd2ckd 192.168.0.99  /z/VOL123  -v VOL123
$ hckd2ckd BIG.ZOS.ADDR:4990 /z/VOL678 -u A8F
$ hckd2ckd 192.168.0.99:4990 /z/host.WORK23 -v WORK23
```

4.4.27 The interrupt command

The **interrupt** command creates an external interruption for a CP.

```
interrupt [cp-number]
```

Where:

`cp-number` is the number of the CP (or zIIP or zAAP or IFL). If not specified, the CP number set by the **cpu** command is used.

The effect of an external interrupt depends on the System z operating system being used. The return values are:

```
0     External interrupt was generated.
12    CP address was not valid.
16    Unable to initialize the manual operations interface.
```

Examples of use are:

```
$ interrupt                (interrupt the default CP)
$ interrupt 1              (interrupt CP number 1)
```

4.4.28 The ipl command

The **ipl** command starts the process of loading an operating system (or a stand-alone utility program).

```
ipl device-number [parm parm-value] [clear]
```

Where:

`device-number` refers to a device number (“address”) in the devmap that contains the initial load program for the operating system.

`parm-value` is a string of up to eight characters that provides additional information for the operating system being loaded.

`clear` causes System z memory to be zeroed before loading the operating system.

The **ipl** function is started on the default CP, which may be set by the **cpu** command. The use of a parm-value completely depends on the operating system being used, and how that operating system has been configured. As a general statement, it is not necessary to **clear** memory before loading an operating system.

The device indicated by the device-number must have *IPL text* as the first record(s). This is normally provided by an operating system utility function. There is a fixed 20 second timeout period for the IPL function to complete, after which a device error message is issued; however, the IPL function continues after the message is issued.

Command return values are:

```
0     IPL function started.
16    Unable to initialize the manual operations interface.
99    The device number is not valid.
```

Examples of command usage are:

```
$ ipl 580
$ ipl 0a80 parm 0a82cs clear
```

4.4.29 The ipl_dvd command

The **ipl_dvd** command emulates IPLing a DVD from the Hardware Management Console (HMC) on a larger System z. The DVD contain files in a unique format for this function to be used. At the time of writing, the only known uses are with an optional form of IBM z/VM system distribution, some Linux for System z distributions, and an older form of z/VSE distribution. zPDT must be operational for this command to be used.

```
ipl_dvd file-name [-q] [-c aaaa       ]
                       [--console aaaa]
```

Where:

`file-name` is the fully qualified name of the `.ins` file on the DVD.

`-q` causes the command to run in quiet mode.

`-c` (or `--console`) specifies the address (device number) of a local 3270 (in the active devmap). This 3270 is then used as an HMC 3270 session. *(At the time of writing this function was not working with z/VM 6.2. Instead, the int3270port function in the [system] stanza of the devmap may be used.)* The use of the -c or --console is deprecated. These options may not be present in future releases.

If `-q` is not specified, the first line of the `.ins` file is displayed and the user is prompted for a continuation signal. The HMC 3270 session has a unique interface that can be used when installing a z/VM system from the standard z/VM distribution DVD.

The return values are:

```
0     Command completed.
8     The .ins file is invalid.
12    The .ins file was not specified.
16    Initialization for manual operation failed, or unable to open .ins file.
```

An example of use is:

```
$ ipl_dvd /media/530_GA_3390_DASD_DVD/cpdvd/530vm.ins
```

4.4.30 The listVtoc command

The **listVtoc** command provides a highly-detailed listing of the VTOC of a CKD volume. It assumes the emulated CKD volume has been initialized with a label and VTOC. This command may be used while zPDT is operational, but it would normally be used when zPDT is not operational. The syntax is:

```
listVtoc ckd-file-name [ckd-file-name] ..
```

Where:

`ckd-file-name is the Linux name of a file containing an emulated 3390 or 3380 volume.`

If all you want are the data set names on the volume, you can pipe the output of **listVtoc** to **grep** to find records containing DSNAME.

Examples of use are:

```
$ listVtoc /z/ZCRES1
$ listVtoc /z/WORK02 | grep -i DSNAME
```

4.4.31 The loadparm command

The **loadparm** command sets an eight-character IPL parameter value that can be read by a special System z instruction. This is also known as a *load* parameter; *IPL* and *load* are used as synonyms in this context.

```
loadparm {value      }
         {-d | display}    (note: there are no minus signs before 'display')
```

Where:

`value` is the character string to be set (up to eight characters).

`-d` or `display` displays the current value.

This value set by this command is available to the operating system during the next IPL. If an IPL parameter is provided as part of an **ipl** command, it overrides any existing loadparm value and is stored as the current **value**. A parameter set this way is maintained only during zPDT operation; it is not retained across multiple zPDT startups.

Return values are:

```
0     The IPL parameter was set or displayed.
16    Unable to initialize the manual operations interface.
```

Examples of command usage are:

```
$ loadparm 0A8200P
$ loadparm -d
```

4.4.32 The managelogs command

The **managelogs** command assists in maintaining summary, trace, and log files in the zPDT logs directory. As a general rule, zPDT maintains these files without assistance, and the **--clean** option of the **awsstart** command can be used to erase all these files. The **managelogs** command is most useful when working with IBM (or a business partner) while investigating a potential zPDT problem. zPDT must not be operational when this command is used.

```
managelogs {file-name       }
           {-s snap-id      }
           {-t date         }
```

Where:

`file-name` removes the summary record and associated file.

`snap-id` removes all summary records and files associated with the specified snap ID.

`date` removes all summary records and files older than the indicated date. The date format is yyy/mm/dd.

The **rassummary** command may be used to determine existing snap ID numbers. There are no return values for this command.

4.4.33 The memld command

The **memld** command is used to write the contents of a Linux file into System z memory, starting at a specified address.

```
memld file-name [address]
```

Where:

`file-name` is a fully qualified Linux file name.

`address` is a System z hexadecimal address. The default is address zero.

Some Linux for System z distributions can be installed by loading various files into System z memory and then executing a System z **restart** command.

Return values are:

```
0     Command complete.
12    File name was not specified.
16    Manual operations initialization failed.
69    The file was not found.
```

An example of the command is:

```
$ memld /tmp/initrd.bin 100000        (meaning address x'100000)
```

4.4.34 The mount_dvd command

The **mount_dvd** command identifies the Linux mount point for a DVD (or CD) that is to be processed as if it were mounted in the DVD drive of a mainframe HMC.

```
mount_dvd complete-path
```

Where:

`complete-path` is the path name to the DVD, but without specifying a particular file name.

This command has a very limited purpose. It is normally used when installing an RSU volume (DVD) associated with z/VM.

An example of the command is:

```
$ mount_dvd /media/zVM_RSU_name/
```

4.4.35 The msgInfo command

The **msgInfo** command provides more information about zPDT messages.

```
msgInfo message-number
```

Where:

`message-number` is the number of a zPDT message.

No return codes are defined for this command. An example of usage is:

```
$ msgInfo AWSCHK208I
AWSINF010I Format:
AWSINF013I     AWSCHK208I Check complete, %d error%s, %d warnings detected.
AWSINF013I
AWSINF011I Description:
```

```
AWSINF013I     The DEVMAP check is complete.
AWSINF013I
AWSINF012I Action:
AWSINF013I     Informational message only. No corrective action needed but
AWSINF013I     if errors are present the DEVMAP cannot be used to start system.
```

All message number are in the form of AWScccnnns where:

```
ccc is the component code issuing the message.
nnn is the message number within the component.
s is the message severity (Debug, Information, Warning, Error, Severe,
Terminal)
```

The message code specified on the **msgInfo** command can omit the AWS prefix and the severity code. For example, `msgInfo chk082` is sufficient. There is also an environment variable named Z1090_MSG to control message formatting.[31] It may be set to FULL (the default), CODE (which will only print the message number and no text), TEXT (which prints the message text and no code) and SHORT (which drops the AWS prefix on the message number).

4.4.36 The oprmsg command

The **oprmsg** command provides input to the System z via the SCLP operator message interface. (This interface is also known as the *HMC console* or the *hardware console*.)

```
oprmsg {text}
```

Where:

`text` is the message to be sent to the System z operating system. If it contains any special characters (such as parentheses), the message should be inclosed in single quotes.

The *hardware console* is used by z/OS if all other consoles fail. It can be used by z/VM, and may be used by Linux for System z. In some cases, the operating system may automatically direct output to the hardware console. In this case, the output will appear in the Linux window where the **awsstart** command was issued. Using an **oprmsg** command from another Linux window may produce confusing results because the response to the command may appear in the original **awsstart** Linux window.

The return values are:

```
0     The message was sent to the SCLP operator interface.
12    No input text was found.
16    Unable to initialize the manual operation interface.
32    Unable to initialize the SCLP message interface.
```

Examples of use are:

```
$ oprmsg 'V CN(*),ACTIVATE'
$ oprmsg 'V 700,CONSOLE'
$ oprmsg 'D A,L'
```

4.4.37 The pdsUtil command

The **pdsUtil** command is a Linux command that reads (or rewrites) members of a z/OS partitioned data set. z/OS is normally not operational when this command is used. The target data set must be a PDS (not PDSE) with FB records. This command cannot change the

[31] This environmental variable could be set with an **export** statement in the Linux shell.

length or number of records in the PDS member. Record length is not limited to 80 bytes. The general operation is to *extract* the PDS member (to Linux), *edit* the Linux file, and then *overlay* the original PDS member with the changed data. Automatic ASCII/EBCDIC translation is provided.

The syntax is:

```
pdsUtil ckd-file-name PDS-name [(mem-name)|/mem-name] [Linux-file-name]

         [-e|-x|--extract]          |
         [-o|--overlay|-r|-replace] |   [-t|--trans|--translate <code>]
         [-l|--list]                |

         [-m|--mbr|--member <mem-name>]
```

Where:

```
ckd-file-name is the Linux name of the file containing the emulated volume.

PDS-name is the z/OS name of the partitioned data set.

mem-name is a member name in the partitioned data set.

Linux-file-name specifies a Linux file to be created (for extract) or written
to the PDS member (for overlay or replace). The default is mem-name.txt.

code is 037/437 or 1047/437 for the code tables to be used for EBCDIC/ASCII
conversion. 037/437 is the default; 1047/437 may work better for international
characters.
```

The PDS member name may be specified in any one of three ways. Using parenthesis around the member name requires that the parenthesis be escaped (so that the Linux shell will not try to process it). If a Linux file name for the member is not specified, the default name is the member name with a .txt suffix. The default name is uppercase or lowercase, depending on how the member name is specified in the command. (The same PDS member is accessed, regardless of case.)

Note that the PDS record length and the number of records in the member cannot change. Only F or FB records may be used. As is implied in the syntax, writing the member back to the PDS performs an update-in-place function.

Examples of usage are:

```
$ pdsUtil /z/WORK02 rb.admin.lib --list                (list the member names)
$ pdsUtil /z/WORK02 rb.admin.lib/ICKDSF --extract      (creates ICKDSF.txt)
$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --extract      (creates ickdsf.txt)
$ gedit ickdsf.text                                    (use Linux editor)
$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --overlay
     (Since no Linux file was named, pdsUtil used ickdsf.txt)
$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --overlay /tmp/myickdsf
     (This is valid, but a dangerous example. The specified Linux file,
      /tmp/myickdsf must be a valid overlay for the target member.)
$ pdsUtil /z/rb.admin.lib\(ickdsf\) --extract      (must "escape" parenthesis)
$ pdsUtil /z/rb/admin/lib --extract --mbr ickdsf    (another way to specify)
```

4.4.38 The query command

The **query** command displays the state of the CPs.

```
query {cp-number   }
      {all         }
```

Where:

`cp-number` is the number of the target CP. The default is the CP number that was set with the **cpu** command.

`all` indicates that the state of all CPs should be displayed.

The return values are:

```
0     Query complete.
12    CP address is not valid.
16    Unable to initialize the manual operation interface.
```

An example of usage is:

```
$ query all
Status for CPU 0 (GP      ,Primary,      Operational): Running
```

The GP in the response indicates a normal CP, as opposed to a zIIP, zAAP, or IFL.

4.4.39 The rassummary command

The **rassummary** command displays information about log and trace files in the ~/z1090/logs directory of the Linux user that started this instance of zPDT. As a general statement, this command is used when working with IBM (or a business partner) while investigating a potential zPDT problem. zPDT need not be running when this command is used.

```
rassummary [-s] [-t] [-d directory-name] [-c comp-name] [-u subcomp-name]

           [-b begin-time] [-e end-time] [-r rec-type]
```

Where:

`-s` indicates only snap records are to be displayed.

`-t` indicates records are to be displayed in chronological order.

`-d` directory-name overrides the normal logs directory name.

`-c comp-name` indicates only records about the indicated component are to be displayed. Component names include the device manager names (in upper case), such as AWSRDR, AWSTAPE, and so forth.

`-u subcomp-name` indicates only records about the indicated subcomponent are to be displayed.

`-b begin-time` indicates only records after the indicated date/time are to be displayed. The format is “yyyy-mm-dd” or “yyyy-mm-dd hh:mm:ss”; these parameters must be enclosed in quotation marks.

`-e end-time` indicates only records before the indicated date/time are to be displayed. The format is the same as for begin-time.

`-r rec-type` indicates that only the specified record type is to be displayed. Valid types are TRACE, LOG, LOG_REGBUF, QD_DUMP, LOG_EVENT, LOG_APPEND, and QUICK_DUMP. Multiple operands may be separated with a comma.

Several options may used to limit the amount of output. IBM service (or a business partner providing zPDT service) will supply component and subcomponent names needed to investigate a problem.

The only documented return value is zero. Examples of command usage are:

```
$ rassummary              (This provides the most general summary)
$ rassummary -r LOG
$ rassummary -r LOG -b”2009-03-03 12:00:00: -e”2009-03-04 23:59:59”
```

4.4.40 The ready command

The **ready** command creates an unsolicited device end interrupt[32] for the indicated device. It is most commonly used with a emulated tape drive to indicate that a new tape volume (which is actually a Linux file) has been mounted or made ready. In some cases **ready** may be useful with an emulated card reader or emulated local 3270 terminal. zPDT must be running in order to use this command.

```
ready device-number
```

Where:

`device-number` is the “address” assigned to the emulated device in the devmap.

The return value is always zero. An example of the command is:

```
$ ready 580           (Device 580 might be an emulated tape drive)
```

4.4.41 The restart command

The **restart** command causes a PSW restart operation on the specified CP.

```
restart [CP-number]
```

Where:

`CP-number` specifies the CP. If this operand is not specified, then the CP number set with the **cpu** command is used.

This command is seldom used. In some cases it may be used to assist an operating system that is stuck in an unusual situation. It is also used to dump a z/VM system and to communicate with some stand-alone utilities.

The return values are:

```
0     The operation is complete.
12    The CP number is not valid.
16    Unable to initialize the manual operation interface
```

Examples of usage are:

```
$ restart             (restart default CP, as set by the cpu command)
$ restart 2
```

4.4.42 The scsi2tape command

The **scsi2tape** command copies a tape volume (mounted on a SCSI tape drive) to a Linux file in awstape format. Linux files in awstape format may be managed and read (by the awstape

[32] Sometimes incorrectly referenced as an *attention* interrupt.

device manager) as though they were tape volumes on a real tape drive. zPDT does not need to be running to use this command.

```
scsi2tape [-c        ][-i      ][-e  nn   ][-s    ] input-dev out-file
          [--compress ][--info  ][--eof nn ][--scan]
                       [-n      ]
                       [--noinfo]
```

Where:

`-c` or `--compress` causes the output awstape file to be written in a compressed format. This is not equivalent to hardware tape compression, such as IDRC.

`-i` or `--info` displays information about each tape file as it is processed. This is the default operation.

`-n` or `--noinfo` suppresses tape file information.

`-e nn` or `--eof nn` specifies the number of consecutive tape marks that indicate the logical end of the tape. The default is two.

`-s` or `--scan` causes the input tape to be scanned, with information displayed (unless `-n` or `-noinfo` is specified). No output file is written.

`input-dev` is the Linux name for the tape drive, such as /dev/st0.

`out-file` is a Linux file name where the awstape formatted file will be written.

In principle, a System z application requiring tape input does not know whether a “real” tape volume (on a SCSI tape drive) or an emulated tape volume (awstape file on an emulated tape drive) is being used. In practice, where repeated mounting and access to the tape may be needed, it may be more convenient to convert the “real” tape volume to an emulated tape volume. Mounting on an emulated tape drive is often much faster than mounting a real tape on a SCSI tape drive.

The optional compression format is unique to zPDT operation. It is not compatible with awstape formats on other platforms and is not related to any type of hardware tape compression. The Linux name of the SCSI tape drive for this command is usually in the /dev/st*n* group and not in the /dev/sg*n* group.

Return values are:

```
0     Function completed without errors.
1     Unable to allocate I/O buffers.
2     Input device not specified, or unable to open input device.
3     Output file not specified, or unable to open output file, or output
      file is write protected.
4     Operation terminated due to an I/O error.
```

Examples of command usage are:

```
$ scsi2tape -n /dev/st0 tape01.awstape
$ scsi2tape -e 4 -s /dev/st0
```

4.4.43 The SecureUpdateUtility command

The **`SecureUpdateUtility`** command is used to manage lease dates in the 1090 token.[33] zPDT should not be running when this command is used. (Note that Linux warning messages are issued a month before the lease date in the token expires.) You must have *root* authority and be in the `/usr/z1090/bin` directory before issuing this command.

[33] Different techniques may be used for updating 1091 tokens.

```
SecureUpdateUtility -r filename
SecureUpdateUtility -u filename
```

The first version (`-r`) writes a request file for the token currently connected to the computer. This request file is unique to the token currently connected.

The second version (`-u`) applies the update file named in the command to the currently connected token. This update file typically extends the *lease date* in the token. The token should be unplugged for at least 10 seconds after an update is applied.

The request file must be sent to a processing facility that can use it to create the update file. The update file is then sent to the user who applies it with the SecureUpdateUtility command. An update file is unique to a token number and may be used only once.

Examples of usage are:

```
$ cd /usr/z1090/bin                 (you must be in this directory)
$ su                                (you must change to root)
# SecureUpdateUtility -r myreq      (creates myreq.req file in Linux)
        (Send the req file for processing; receive an upw file in return)
# SecureUpdateUtility -u myreq.upw   (apply the update file)
# exit
```

4.4.44 The SecureUpdate_authority command

The **SecureUpdate_authority** command adds a Linux userid or removes a Linux userid from a list of userids that may issue the **zpdtSecureUpdate** command, which executes the **SecureUpdateUtility** command internally, without requiring the user to operate as *root* and be positioned in the /usr/z1090/bin directory. The **SecureUpdate_authority** and **zpdtSecureUpdate** commands allow the installation to avoid usage of *root* when updating token licenses. The **SecureUpdate_authority** command must be run as *root*, but it is used only once for a given userid.

```
SecureUpdate_authority [-a | -d] userid

-a adds the indicated userid to the list of userids that are allowed to issue
   the SecureUpdateUtiluty command.
-d removes the indicated userid from this list.
```

A command example is:

```
# SecureUpdate_authority -a ibmsys1       (issued by root)

$ zpdtSecureUpdate -r myreq              (issued by userid ibmsys1)
```

Once a userid is authorized to use **zpdtSecureUpdate**, it is no longer required to change to the `/usr/z1090/bin` directory. The availability of these commands does not preclude direct usage of **SecureUpdateUtility** by a *root* user.

4.4.45 The senderrdata command

The **senderrdata** command packages zPDT diagnostic information and, optionally, sends the package to IBM. zPDT need not be running when this command is used.

```
senderrdata
```

There are no operands. The command produces menus and prompts; the initial menu is:

```
              z1090 Error Data Processing Script
Options:
   1  rassummary         execute the rassummary command
   2  rassummary -s      execute rassummary -s
   3  FTP/dump snapdata data
   4  FTP/dump PE directed data
   5  Create configuration information file
   6  Logs directory maintenance
   7  FTP/dump rassummary created files
   8  FTP/dump all files in log directory
   9  snapdump
```

The FTP/dump function provided in several of the options means that information can be sent (via FTP) to the IBM *test case* site or it can be retained in a local Linux *dump* file (which is a zipped tar file). Data should not be sent to IBM unless a problem record has been opened by the business partner who provided the zPDT system. The business partner can provide assistance in using the various **senderrdata** options and parameters.

There are no defined return values for this command.

4.4.46 The settod command

The **settod** command sets the specified time/date in the System z Time Of Day (TOD) clock during the next IPL of an active zPDT system. The TOD change is not carried across restarts of zPDT. When used, this command would normally be issued after the **awsstart** command and before an **ipl** command. zPDT normally sets the emulated System z TOD clock to match the underlying PC TOD clock; this command alters that normal action. A **settod** command issued while a System z operating system is active has no immediate effect; it takes effect only during a subsequent **ipl** command.

The full syntax is:

```
settod YYYY/MM/DD-HH:MM:SS
settod YYYY/MM/DD
settod HH:MM:SS                (the :SS portion may be omitted)
```

If both date and time are present, they must be separated with a dash without blanks between the elements. A time value is expressed in 24-hour notation. The output of the command shows the adjustment that is made to the default TOD value. The minimum YYYY value is 1900.

This command does not change the Linux hardware clock value in any way and does not affect the time stamps that are stored in the zPDT token. This command provides a way to test System z software operation at future times (or past times). After the subsequent **ipl**, the System z TOD clock is incremented in the normal way, starting at the time/date specified in the **settod** command.

Assume the current date and time (in the PC hardware clock) is July 20, 2013 at 1 PM:

```
$ settod 16:40                    (July 20, 2013, 4:40 PM)
$ settod 2012/7/20                (July 20, 2012, 1 PM)
$ settod 2005/1/1-00:00           (January 1, 2005, midnight)
```

In principle, any portion of the parameter that is omitted is assumed to be the same as the TOD value in the base Linux system. The date field is processed right to left and the time field is processed left to right. If a single number with no delimiters is used as the parameter it is assumed to be a day number. However, we suggest you always enter a full date or time or both.

4.4.47 The snapdump command

The **snapdump** command causes various diagnostic data and logs to be collected and written in the ~/z1090/logs directory. zPDT must be running when this command is used. This command may be used when a zPDT problem situation exists while zPDT is running.

```
snapdump [-c comp-name[subcomp-name]][-d “desc-text”]
```

Where:

`comp-name` is a component name; only information related to this component is obtained. Component names include device manager names (in upper case).

`subcomp-name` is a subcomponent name; only information related to this subcomponent is obtained. The component name must also be specified.

`desc-text` is descriptive text (in quotes).

If no options are specified, then information about all active components and subcomponents is collected.

zPDT automatically collects diagnostic information when a zPDT failure occurs. The **snapdump** command is intended only for situations where the user observes a zPDT failure or problem that is not detected by zPDT itself. This command is not useful for System z operating system problems or problems with the underlying Linux system.

No return values are defined. An example of the command is:

```
$ snapdump -d “This is a test”
```

4.4.48 The st command

The **st** (store) command is used to alter registers or memory in a CP (or zIIP or zAAP or IFL). The syntax is similar to that for the **d** command.

```
st {p xxx xxx xxx xxx  (expressed as 4 words)           }
   {pfx xxx                                             }
   {gn xxx      (32 bit register usage)                 }
   {gxn xxx     (64 bit register usage)                 }
   {yn xxx                                              }
   {xn  xxx     (32-bit usage)                          }
   {xxn xxx     (64-bit usage)                          }
   {zn xxx                                              }
   {hex-addr xxx                                        }
```

Where:

`xxx` is a hexadecimal value to be stored.

`p` alters the current PSW.

`pfx` alters the prefix register.

`g`*n* alters the contents of a general purpose register. A maximum of a 32-bit operand can be specified.

`gx`*n* alters the contents of a general purpose register. Up to 64 bits may be specified.

`y`*n* alters the contents of a floating point register.

`x`*n* or `xx`*n* alters a control register. The first format uses a 32-bit operand and the second format uses a 64-bit operand.

`z`*n* alters an access register.

`hex-addr` is an absolute address in memory.

Only real memory (as opposed to virtual memory) can be addressed by this command. Memory is altered byte-by-byte, to match the operand. It is possible to display a virtual address (with the **d** command), note the real address of the page that is displayed, and then use the **st** command to modify memory in the real page. The CP should be in a stopped state before any of these alter functions are used.

The return values are:

```
0     Command complete.
-2    No arguments specified.
```

Examples of use are:

```
$ st p FF007AB0 0 0 123456  (set 64-bit PSW)
$ st g2 123           (change low-order 32 bits of GPR2 to x'00000123')
$ st gx2 123          (change 64 bits of GPR2 to x'0000000000000123')
$ st 461244 32        (change byte at real address x'461244' to x'32')
```

4.4.49 The start command

The **start** command starts a CP that was in the stopped state (due to a prior **stop** command).

```
start [CP-number]
```

Where:

`CP-number` is the target CP number. If this operand is not specified, the CP number set by the **cpu** command is used. The constant **all** may be used instead of a cpu number.

The return values are:

```
0     Operation complete.
12    CP number is invalid.
16    Unable to initialize the manual operation interface.
```

Command examples are:

```
$ start
$ start 1
$ start all
```

4.4.50 The stop command

The **stop** command places a CP in the stopped state. It may be restarted with a **start** command or a reset function.

```
stop [CP-number]
     [all      ]
```

Where:

`CP-number` is the target CP number. If this operand is not specified, the CP number set by the **cpu** command is used. The constant **all** may be used instead of a cpu number.

Generally, a CP is stopped in order to display register or memory contents. In rare cases, it might be stopped to halt the process of an application or operating system function.

The return values are:

```
0     Operation complete.
12    CP number is invalid.
16    Unable to initialize the manual operation interface.
```

Command examples are:

```
$ stop
$ stop 1
$ stop all
```

4.4.51 The storestatus command

The **storestatus** command causes certain CP registers to be stored in fixed memory locations, as defined in z/Architecture. This command is typically used before taking a standalone dump.[34]

```
storestatus [CP-number]
```

Where:

`CP-number` specifies the target CP. If this operand is not specified, then the CP indicated by the **cpu** command is used.

The CP must be in the stopped state (via a **stop** command) when **storestatus** is issued.

There is no defined return code for this command. An example of the command is:

```
$ stop all
$ storestatus
```

4.4.52 The storestop command

The **storestop** command places the CP in a stopped state if memory at the indicated address is altered. A **start** command may be used to resume execution; the storage operation will be completed

```
storestop hex-address [off | OFF]
```

Where:

`hex-address` is a memory address.

`off` or `OFF` removes an existing storestop function.

The target address is the effective address, which is typically a virtual address but could be a real address when in DAT-off mode. Only one storestop address can be in use. A subsequent storestop changes the address being monitored. The memory alteration is completed before stop state is entered. A **start** command may be used to resume program execution.

The target address could be in any address space. This command is not intended for routine application debugging. Note that if multiple CPs are in use, the **storestop** may need to be set for each CP.

[34] Recent z/OS and System z usage have removed the need for this command. The System z (including zPDT) is primed by z/OS to perform a store status command before the next IPL.

Return values are:

```
0     The address stop was set.
16    Unable to initialize the manual command interface.
69    The hex address is not valid.
101   No storestop address is set.
```

An example of usage is:

```
$ storestop 4FCC
```

4.4.53 The sys_reset command

The **sys_reset** command performs a system reset (or system reset clear) function as defined by z/Architecture.

```
sys_reset [normal | clear]
```

Where:

`normal` is the default operation.

`clear` performs the additional architected clear function.

No return values are defined for this command. An example of usage is:

```
$ sys_reset
```

4.4.54 The tape2file command

The **tape2file** command reads a file from an emulated tape volume (awstape format) and writes a simple Linux file. The various awstape control bytes (in the input file) are removed before writing the output file. The result is a simple string of bytes in the output file, with no indication of the separation of blocks that existed on the input tape. zPDT need not be operational to use this command.

```
tape2file [-f file-num] input-file output-file
```

Where:

`-f file-num` specifies the logical file number in the input file. The default is file 0 (the first file). A logical tape mark separates files.

`input-file` is the name of a Linux file that is in awstape format.

`output-file` is the name of a Linux file for output.

The output file is a binary file; it is possible that the System z application included NL separators that would indicate a Linux text file, but this is up to the System z application. System z tape labels (in the input file) are not recognized and are treated simply as data files.

No return values are defined. An example of the command is:

```
$ tape2file -f 2 /z/111111.awstape /tmp/mine/testfile
```

4.4.55 The tape2scsi command

The **tape2scsi** command copies a logical tape volume (in awstape format) to a SCSI tape drive. The output tape is blocked as indicated by the control bytes within the awstape format. zPDT does not need to be running to use this command.

```
tape2scsi [-i      ] [-e  nn   ] [-s    ] input-file out-dev
          [--info  ] [--eof nn ] [--scan]
          [-n      ]
          [--noinfo]
```

Where:

`-i` or `--info` displays information about each tape file as it is processed. This is the default operation.

`-n` or `--noinfo` suppresses tape file information.

`-e nn` or `--eof nn` specifies the number of consecutive tape marks that indicate the logical end of the input tape. The default is two.

`-s` or `--scan` causes the input file to be scanned, with information displayed (unless `-n` or `-noinfo` is specified). No output tape is written.

`input-file` is a Linux file name with data in awstape format.

`out-dev` is the Linux name for the tape drive, such as `/dev/st0`.

This command converts a logical tape volume to a real tape volume. The optional compression format used by zPDT awstape functions is recognized and processed, if present. The Linux name of the SCSI tape drive for this command is usually in the /dev/st*n* group and not in the /dev/sg*n* group.

Return values are:

```
0     Function completed without errors.
1     Unable to allocate I/O buffers.
2     Input file not specified, or unable to open input file.
3     Output device not specified, or unable to open output file, or output
      device is write protected.
4     Operation terminated due to an I/O error.
```

An example of command usage is:

```
$ tape2scsi -n  tape01.awstape /dev/st0
```

4.4.56 The tape2tape command

The **`tape2tape`** command copies a logical tape volume (in awstape format) to another logical tape volume (also in awstape format). Several optional operations may take place during the copy, including compressing or uncompressing the data. The primary purpose of the command is to compress and uncompressed awstape file (or vice versa), or to summarize a file. zPDT does not need to be running to use this command.

```
tape2tape [-c        ][-d        ][-e nn   ][-i      ]
          [--compress][--dynainfo][--eof nn][--info  ]
                                            [-n      ]
                                            [--noinfo]

          [-s    ] in-file out-file
          [--scan]
```

Where:

`-c` or `--compress` causes the output tape to be compressed.

`-d` or `--dynainfo` displays tape content when each record is read. Otherwise, information is displayed only when a tape mark is encountered.

-e nn or --eof nn specifies the number of consecutive tape marks that indicate the logical end of the input file. The default is two tape marks.

-i or --info provides a summary of the tape volume. This is the default.

-n or --noinfo indicates no summary is to be displayed.

-s or --scan scans the tape, but no output is produced.

in-file is the name of a Linux file in awstape format.

out-file is the name of a Linux file that will be in awstape format.

The input file may be in the zPDT compressed format; this is handled automatically. The output file is compressed only if that option is selected. Both input and output files are in awstape format. This command cannot convert other Linux files to awstape format.

No return values are defined for this command. An example of the command is:

```
$ tape2tape -e 1 /tmp/111111  /z/222222.awstape
```

4.4.57 The tapeCheck command

The **tapeCheck** command verifies the internal format of a logical tape volume in awstape format. That is, it verifies that the awstape control bytes within the file are logically correct. zPDT does not need to be running to use this command.

```
tapeCheck file-name
```

Where:

file-name is a Linux file in awstape format.

This command is used to inspect awstape files that may have been corrupted. It could be used to check awstape files generated on another platform, to ensure they are compatible with zPDT. Please note that some older platforms do not create correct awstape bytes at the end of a logical tape volume.

The return value is equal to the number of errors found in the awstape format. An example of the command is:

```
$ tapeCheck /tmp/222222.awstape
```

4.4.58 The tapePrint command

The **tapePrint** command writes the content of an emulated tape volume (awstape format) to Linux stdout. zPDT does not need to be running to use this command.

```
tapePrint [-a     ][-e      ] in-file
          [--ascii][--ebcdic]
```

Where:

-a or --ascii specifies that the emulated tape volume has ASCII characters.

-e or --ebcdic specifies that the emulated tape volume has EBCDIC characters. This is the default format.

in-file specifies a Linux file that is in awstape format.

Output is displayed block by block, in both hexadecimal and character format.

No return values are defined for this command. An example of the command is:

```
$ tapePrint /z/222222.awstape
```

4.4.59 The token command

The **token** command displays the characteristics of the zPDT token currently in use. This command should be used when zPDT is running.

```
token
```

There are no operands. Only the number of token licenses currently in use are displayed. That is, if the token allows three CPs, but only one CP is currently in use, then information for only one CP is displayed. The only return value defined is zero. An example of command use is:

```
$ token
CPU 0, zPDTA ... Serial 6186(0x182A) Lic=88570(0x159FA) EXP=4/15/2013 1090
```

The serial number is the effective System z serial number and may differ from the token serial number. The license serial number reflects the token serial number used to provide the zPDT license. The final output indicator is 1090 or 1091 where 1090 indicates the original zPDT version and 1091 indicates the RDz version.

4.4.60 The txt2card command

The **txt2card** command reads a Linux text file and creates a card image file (in EBCDIC).

```
txt2card in-file out-file
```

Where:

`in-file` is the name of a Linux text file (in ASCII). Each record must be 80 bytes or shorter.

`out-file` is the name of a Linux binary file that is written by this command.

Input records are extended (with blanks) to 80 bytes and then converted to EBCDIC. The ASCII/EBCDIC conversion table is fixed and cannot be customized.

There are no defined return values for this command. A command example is:

```
$ txt2card /tmp/work2/config.txt /z/cards/deck1
```

4.4.61 The uimcheck command

The **uimcheck** command displays the state of the Unique Identity Manager (UIM) serial number (which is the System z serial number). zPDT need not be running when this command is issued; any user may issue the command.

```
uimcheck
```

There are no operands.

4.4.62 The uimreset command

The **uimreset** command is used to reset (remove) the UIM serial number from either the local UIM database or both the local and remote UIM databases. This command must be run as *root*, and zPDT is normally not running when this command is used.

```
uimreset [-l] [-r]

-l indicates that the UIM serial number in the local UIM database should be
   erased.
-r indicates that the UIM serial number in both the local UIM database and
   the remote UIM server should be erased.
```

The Unique Identity Manager (UIM) function and its usage are explained in detail in the Chapter titled “License & serial number servers” in the third volume of this documentation series.

4.4.63 The uimserverstart command

The **uimserverstart** command is used to start a remote UIM server. This command should always be issued by the same Linux userid because it saves a database in the home directory of that Linux user. It must not be started by *root*.

```
uimserverstart
```

This command also adds the UIM server to the cron lists of the Linux system. This causes the UIM server to be restarted (if it fails) and to be automatically started when that Linux system is rebooted.

A UIM server is normally used only as part of a remote zPDT license server environment. It is not used when running a simple zPDT environment with a token connected to the local zPDT system.

The Unique Identity Manager (UIM) function and its usage are explained in detail in the Chapter titled “License & serial number servers” in the third volume of this documentation series.

4.4.64 The uimserverstop command

The **uimserverstop** command causes a running UIM server to be stopped. The cron entries that automatically start the UIM server are also removed. This command is not used as *root*.

```
uimserverstop
```

4.4.65 The z1090instcheck command

The **z1090instcheck** command checks a number of installation criteria. In may be used whether or not zPDT is running.

```
z1090instcheck
```

There are no operands. This command is sensitive to the Linux distribution being used and to the level of that distribution. The output may vary with a new release of zPDT. The **z1090instcheck** command is used with both 1090 and 1091 systems.

Return values are:

```
0    Command completed.
8    An unrecognized Linux system is being used.
9    Only root can use this command.
```

An example of usage is:

```
$ su             (change to root)
$ z1090instcheck
1. SUSE at version 10.3 which is                      OK
2. SUSE kernel.shmmax is 2415919104 which is          OK
3. SUSE kernel.msgmni is 512 which is                 OK
4. SUSE kernel.core_uses_pid is 1 which is            OK
5. SUSE kernel.core_pattern is Core-%e-%p-%t which is OK
6. SUSE unlimited ic is set to unlimited which is     OK
and so forth
```

Remember that the specific report changes with new zPDT releases and with the underlying Linux distribution. Some of the checks may produce warnings that you must evaluate for yourself.

4.4.66 The z1090term command

The **z1090term** command provides an ASCII terminal function that may be used to connect to the HMC-like ASCII terminal defined by the intASCIIport statement in a zPDT devmap. The syntax is:

```
z1090term IPaddress:port
```

where the IPaddress points to the zPDT Linux base and the port number is defined in the intASCIIport statement of the zPDT device map. Examples are:

```
z1090term my.remote.zpdt.com:7100
z1090term 192.168.1.101:7100
z1090term localhost:7100
```

There is no standard port number for this function; the 7100 number in the examples is arbitrary. This command is included in the zPDT libraries.

4.4.67 The z1090ver and z1091ver command

The **z1090ver** or **z1091ver** command displays the current zPDT version, with the date it was build. This information may be necessary when investigating a zPDT problem.

```
z1090ver
```

There are no operands. The return value is zero. An example of the command is:

```
$ z1090ver
z1090, version z1090_v1r0_E39, build date - 10/17/08 SUSE 32 bit
```

The exact output messages vary with the zPDT release.

4.4.68 The zpdtSecureUpdate command

The **zpdtSecureUpdate** command allows token license update functions without being required to operate as root. Before a userid can issue this command, that userid must be added to a list of userids that are authorized to use the **zpdtSecureUpdate** command. This list is managed by the **SecureUpdate_authority** command.

```
zpdtSecureUpdate [-r | -u] Linux-file-name

The operands (-r, -u, and file-name) are the same operands used with the
SecureUpdateUtility command. See that command for details.
```

4.4.69 Command summary

Some zPDT commands must be used when zPDT is active, some cannot be used when zPDT is active, and some do not care whether zPDT is active. In Table 4-2, the following are shown:

- Y indicates that zPDT must be operational to use the command.
- N indicates that zPDT cannot or should not be operational when the command is used.
- E indicates that zPDT can be either operational or not operational when the command is used.

Table 4-2 Command environments

Command	Use	Command	Use	Command	Use
adstop	Y	ipl	Y	stop	Y
alcckd	E	ipl_dvd	Y	storestatus	Y
alcfba	E	loadparm	Y	storestop	Y
awsckmap	E	managelogs	N	sys_reset	Y
awsin	Y	memld	Y	tape2file	E
awsmount	Y	msgInfo	E	tape2scsi	E
awsstart	N	oprmsg	Y	tape2tape	E
awsstop	Y	query	Y	tapeCheck	E
card2tape	E	rassummary	E	tapePrint	E
card2txt	E	ready	Y	token	E
ckdPrint	E	restart	Y	txt2card	E
cpu	Y	scsi2tape	E	z1090instcheck	E
d	Y	senderrdata	E	hckd2ckd	E
fbaPrint	E	snapdump	Y	SecureUpdateUtility	N
find_io	E[a]	st	Y	ap_ (several commands)	Y
interrupt	Y	start	Y	settod	Y
mount_dvd	Y	hfba2fba	E	htape2tape	E
pdsUtil	E[b]	listVtoc	E[b]	clientconfig	N
uimreset	N	uimserverstart	N	uimserverstop	N
SecureUpdate_authority	E	clientconfig_authority	E	zpdtSecureUpdate	N
aws_sysctl	N	aws_bashrc	N	z1090term	

a. In the case of tunnel OSA connections, the `find_io` command might indicate different results before zPDT is first started.

b. In general, we suggest that these commands not be used when zPDT is active, although there is not an absolute rule for this.

5

Frequently asked questions

The following FAQs are typically asked when first studying zPDT. Volume 2 of this book series contains additional FAQs that are more specific to installation and detailed usage.

Q: Is this a multi-user system?
A: Yes. Multiple TSO users can connect, in several ways, and use the system in the normal manner. The same applies to z/VM users, CICS users, and so forth.

Q: How many users can the system support?
A: There is no definitive answer to this. The aws3274 device manager supports 32 connections (which emulate local 3270 devices). There is no specific maximum for TCP/IP (awsosa) connections to z/OS or for SNA connections[1]. Practical performance is the primary limitation, not the theoretical connectivity for terminal connections. A given system might do well for one very heavy DB2 user, or 10-30 typical TSO users, or 100 web users each having a low transaction rate to HTTPD. The answer to the question depends completely on the nature of the workloads involved.

Q: Do the int3290port or intASCIIport interfaces provide the same functions as the “HMC console” in a larger System z?
A: No. An HMC (on a larger System z) provides the HMC console function and also provides a 3270 terminal and an ASCII terminal. These are three separate interfaces. Output sent to the “HMC console” (on a zPDT system) appears in the Linux window that issued the `awsstart` command. Input to the HMC console is through the zPDT`oprmsg` command.

Q: The new --interface parameter for awsosa is confusing. How should I use it?
A: Read Chapter 10 in the third book (SG24-7723-04 or later). The --interface parameter was introduced because recent Linux distributions have changed the way LAN interfaces are named and a more general method of specifying a LAN interface to awsosa was needed.

Q: Will my devmap from a previous zPDT release work with the new release?
A: Maybe. An important issue is with path names for OSA interfaces. Previous zPDT releases considered only LAN interfaces that were not *down* when assigning path names. The current release considers all detected LAN interfaces and this may result in a different path name for OSA.

[1] SNA usage is not supported on the 1090 a this time.

Q: Most of the documentation is about large ThinkPads or servers. I have a typical desktop PC. Can I use zPDT with it?
A: Probably, assuming Linux works properly with the display, DVD drive, power, and LAN interfaces on your desktop. You should have *at least* 4 GB of memory (more is better). However, the only formal support is for the machines described in this document. IBM simply cannot undertake the extensive testing that would be needed to qualify the vast variety of PCs that exist. Note that hiperthreading must be disabled.

Q: Can I move a zPDT token between two PCs?
A: Technically, yes,[2] but there is an important issue involved. The latest time-of-day value seen by the underlying PC hardware is stored in the token. If the token then encounters an earlier time, it will fail the operation with a *time cheat* message. If your two PCs have a significant time spread between their hardware time-of-day clocks, you may have problems.

Q: What are the maximum numbers of CPs, zPDT instances, and I/O devices?
A: A maximum of 8 CPs (or combinations of CPs, zIIPs, zAAPs, and IFLs) may be used in a zPDT instance, although your license terms may have a lower limit. A maximum of 15 zPDT instances may exist in a Linux system. A maximum of 1024 I/O devices may be defined in an instance. *Do not* take these program maximum values as being practical environments. There are other factors (such as available memory, SMP effects, and I/O capability) that limit practical usage.

Q: Does zPDT reserve PC core(s) for System z execution? Does it partition PC memory in some way to create System z memory?
A: The answer is No to both questions. A running zPDT system consists of many processes and threads under Linux; these are dispatched in the normal Linux manner and reference Linux virtual memory in the normal way.

Q: Some of the ThinkPads that you discuss have two or four processors. Are all used for zPDT operation?
A: Partly. For a 1090 L01 model, only one PC processor is used for System z instruction execution, which is a single Linux *process*. However, the other processor may be used to help prepare instructions for the primary 1090 processor. The other processor may also be used for other processes, including I/O activity by the 1090. Exactly which processor is used for which purpose at any instant depends on normal Linux dispatching. Only one zPDT CP should be used with a two-core PC.

Q: Are two or more processor cores needed? Can I use a PC with a single core?
A: Two processor cores are not required for an L01 system. Working with a single core simply results in a slower system because the single processor must handle all System z CP operations plus all the other processes for I/O and other Linux details. We recommend using a system with at least two PC processors (“cores”). (The one-core machine is an exception to the rule that there must be at least one more core than the number of CPs.)

Q: Can I use a USB disk drive for emulated System z data?
A: Yes, assuming the base Linux recognizes and supports the drive in the normal manner. It may offer slightly less performance than the internal PC disk drives, but this may be acceptable in many cases.

Q: Can I use another hard disk in the Ultrabay slot of my laptop?
A: Yes, assuming the base Linux recognizes and supports it. However, remember that you may need a CD/DVD drive to install System z software and you probably have only one Ultrabay slot. You might consider using a docking station to obtain another Ultrabay slot. Another option is to use a USB-attached CD/DVD drive.

[2] You should, of course, observe the terms and conditions of your zPDT license agreement.

Q: Can I use a USB-attached CD/DVD drive?
A: Yes, assuming Linux recognizes it correctly. In some cases we noticed that these were *much* slower than the internal CD/DVD drive, which is relevant when loading a complete z/OS system.

Q: Should I use AHCI or compatibility mode for the laptop disk?
A: Linux seems to install correctly either way. However, we have reports that setting AHCI (in BIOS) instead of Compatibility mode greatly improves performance of Ultrabay disks, but we do not have more exact information about specific configurations. zPDT does not care about these settings; it simply uses Linux I/O functions.

Q: Does more PC memory improve performance?
A: Yes. Linux can effectively use memory as a disk cache and this enhances performance.

Q: Is there an adapter for parallel channels?
A: There are currently no hardware channel adapters for zPDT systems.

Q: I already have existing ESCON® adapters for my Intel base PC. Can I use these?
A: No.

Q: Can I use VMWare or another virtual manager to place Windows and Linux on the same machine?
A: See the third document in this series (SG24-7723) for a discussion of supported virtual environments.

Q: Can I use a dual boot method to place Windows and Linux on the same machine?
A: Yes, provided you have sufficient disk space. The primary challenge may be to prevent Linux or Windows updates from overwriting the dual boot functions.

Q: I want to evaluate a System z configuration with a zIIP using my two-core laptop. How do I configure this?
A: You could accomplish your goal by using z/VM to provide simulated zIIPs and zAAPs. You should not directly configure both a zPDT CP and a zIIP on your two-core machine.

Q: I am short on USB ports. Can I use a USB extender for the token connection?
A: Do not use an *unpowered* USB port extender; it may render your zPDT token unusable. A powered USB port extender should work correctly.

Q: Are the new System z instructions (as provided with recent new System z machines) present?
A: Yes, up through the EC 12 level. A few instructions dealing with functions not present in a zPDT environment are not available.

Q: Can `awstape` files from P/390 or R/390 systems be used with the zPDT offering?
A: In general, yes. There is a restriction that the P/390 or R/390 `awstape` file cannot be read beyond the last valid logical data record. The older awstape files do not contain the proper indicators after the last logical data record. (However, awstape files created by zPDT work correctly in this situation.) This situation is typically encountered when using "tape map" programs that attempt to read everything on a tape, without obeying the normal EOV/EOF records or double tape marks normally used to indicate the logical end of data on a tape.

Q: Can awstape files from Multiprise 3000 systems and FLEX-*ES* systems be processed? Can awstape from zPDT be sent to these systems?
A: Yes, and yes.

Q: Can emulated CKD files (for 3390s, for example) be copied from P/390, R/390, Multiprise 3000, or FLEX-*ES* systems?

A: In general, no; the internal formats are slightly different. These are typically transferred by dumping the drive (with ADRDSSU, DDR, or something similar) to an awstape file, moving the awstape file, and restoring it on the target system. The awstape files provide the mechanism that is compatible among these systems.

Q: Can I use my "brand X" TN3270e client?
A: Maybe, but do not base any error reports to IBM on it. Not all TN3270e clients are the same and there can be significant differences in the handling of error and recovery situations. (This is especially relevant to the z/OS console, which uses unusual CCW sequences to provide something like full duplex operation.)

Q: Is Flashcube supported for emulated disks?
A: No.

Q: Why do you support only limited Linux releases?
A: IBM performs very extensive testing for zPDT. We use Linux releases that are current at the time this testing starts. There are many practical reasons for not changing the Linux level midway in the testing cycles.

Q: Will using a zIIP or zAAP increase the performance of my 1090?
A: No, assuming you are replacing a CP with the zIIP or zAAP. These speciality processors operate at the same speed as a "normal" 1090 CP. They are provided to allow developers to verify that their applications use a zIIP/zAAP in the intended manner. Of course, the use of a zIIP or zAAP might allow more parallel operation in your workload, which could increase the performance under zPDT.

Q: Is the OSA function for ICC provided?
A: No. However, the AWS3274 device manager provides approximately the same service.

Q: I sometimes want to change Linux TCP/IP between DHCP and a static IP address. Can I do this while the 1090 is running? I am changing only Linux parameters, not OSA parameters.
A: This is not supported, not tested, and probably will not work correctly. We suggest you do not change Linux LAN definitions while the 1090 is running if you are using OSA functions.

Q: Is Token Ring supported for emulated OSA usage?
A: No.

Q: Can I use the emulated OSA QDIO with IPv6?
A: Yes. You can also use it for aws3274 clients if you find a client (and Linux host) that supports IPV6.

Q: Do I need to change any z/OS parameters to operate with zPDT?
A: In principle, no. In practice, you may need to adjust a few parameters. These are primarily related to performance. For example, the CICS transaction timeout value might need to be increased for very "heavy" transactions.

Q: I have zPDT and a hardware key. Where can I download z/OS?
A: z/OS (or any other IBM System z software) is not part of the base zPDT product. You need to discuss this question with your zPDT source.

Q: Can I use MVS 3.8?
A: No. The 1090 does not support architectures prior to XA and 3380/3390s.

Q: All your examples have three-digit emulated device addresses. Is this required?
A: No, you may use three- or four-digit addresses. The use of only three-digit addresses with the AD-CD z/OS system is a historical accident.

Q: Do I need the hardware token to *install* zPDT?
A: No, you need it only to *run* zPDT.

Q: Can zPDT support older CKD drives, such as 3350s?
A: No.

Q: Can I use ICKDSF with the ANALYZE function for emulated CKD volumes?
A: No, in most cases. Emulated CKD devices (such as 3390s) do not contain spare cylinders and diagnostic cylinders that may be required for ANALYZE operation.

Q: How many emulated devices can I have?
A: The architected maximum for zPDT is about 64 K, but lower limits are set in each 1090 release. Each device is seen as a *subchannel*. Allocated subchannels take memory space in 1090 control blocks; an excessively large number of these control blocks can impact performance through memory usage and list search times. The current zPDT release for Linux has a limit of 1024 subchannels.

Q: Is there an IBM program number associated with the core zPDT software?
A: Yes, it is 5799-ADE, which is a PRPQ. In the normal course of events, zPDT users do not need to deal with this number.

Q: How accurate are the System z TOD and timer functions?
A: To a large extent, these are approximately as accurate as the timer in the underlying PC. Some interval measurements may have a granularity of about 500 microseconds (plus the System z operating system time needed to manage time-related activities).

Q: I need to have multiple levels (often more then 3) of z/OS available for testing, although each z/OS is usually idle at any given time. A 1090-L03 seems to be overkill for my modest processing needs and, in any case, is limited to three instances. How can I address this problem?
A: The easiest solution is to use z/VM with multiple z/OS guests. This requires some z/VM skills, but these are relatively modest. It probably requires more System z memory than other potential solutions, to avoid excessive z/VM and z/OS paging. See the note in "System stanza" on page 34 about older z/OS releases.

Q: Are the zIIP, zAAP, or IFL processors faster than the default CP? Why would these be used?
A: They are not faster. They can be used to verify that you code is using the speciality processors. Also, a zIIP and zAAP runs in parallel with the CP function that dispatched it and may offer improved performance in this way. An IFL might be used simply to verify that code (such as Linux for System z, or a basic z/VM) does run correctly in an IFL.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 91. Note that some of the documents referenced here may be available in softcopy only.

- *IBM System z Personal Development Tool Volume 2 Installation and Use*, SG24-7722
- *IBM System z Personal Development Tool Volume 3 Additional Topics*, SG24-7723
- *IBM System z Personal Development Tool Volume 4 Coupling and Parallel Sysplex*, SG24-7859

Other publications

These publications are also relevant as further information sources:

- *z/Architecture Principles of Operation,* SA22-7832
- *System z Personal Development Tool User’s Guide and Reference*, G229-1101

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM Redbooks

IBM System z Personal Development Tool: Volume 1 Introduction and Reference

(0.2”spine)
0.17”<->0.473”
90<->249 pages

IBM System z Personal Development Tool
Volume 1 Introduction and Reference

System z Development Tool

Full z/OS usage

Linux base

This IBM Redbooks publication introduces the IBM System z Personal Development Tool (zPDT), which runs on an underlying Linux system based on an Intel processor. zPDT provides a System z system on a PC capable of running current System z operating systems, including emulation of selected System z I/O devices and control units. It is intended as a development, demonstration, and learning platform and is not designed as a production system.

This book, providing an introduction, is the first of three volumes. The second volume describes the installation of zPDT (including the underlying Linux, and a particular z/OS® distribution) and basic usage patterns. The third volume discusses more advanced topics that may not interest all zPDT users. The IBM order numbers for the three volumes are SG24-7721, SG24-7722, and SG24-7723. An additional volume (SG24-7859) describes the use of zPDT in a Parallel Sysplex configuration.

The systems discussed in these volumes are complex, with elements of Linux (for the underlying PC machine), z/Architecture (for the core zPDT elements), System z I/O functions (for emulated I/O devices), and z/OS (providing the System z application interface), and possibly with other System z operating systems. We assume the reader is familiar with general concepts and terminology of System z hardware and software elements and with basic PC Linux characteristics.

SG24-7721-05 ISBN 0738438294