ibm.com/redbooks

ISV IBM zPDT Guide and Reference

Bill Ogden

z Systems

IBM Redbooks

ISV IBM zPDT Guide and Reference

December 2022

SG24-8205-06

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

Seventh Edition (December 2022)

This edition applies to Version 1, Release 11 of Independent Software Vendor IBM Z Personal Development Tool (IBM zPDT).

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication provides both introductory information and technical details for ISV IBM Z® Program Development Tool (IBM zPDT®), which produces a small IBM zSystems environment that is suitable for application development. ISV zPDT is a personal computer (PC) Linux application. When ISV zPDT is installed on Linux, normal IBM zSystems operating systems (such as IBM z/OS®) may be run on it. ISV zPDT provides the basic IBM zSystems architecture and provides emulated IBM 3390 disk drives, 3270 interfaces, Open Systems Adapter (OSA) interfaces, and other items.

The systems that are described in this publication are complex, with elements of Linux (for the underlying PC machine), IBM z/Architecture® (for the core zPDT elements), IBM zSystems I/O functions (for emulated I/O devices), z/OS (the most common IBM zSystems operating system), and various applications and subsystems under z/OS. We assume that the reader is familiar with general concepts and terminology of IBM zSystems hardware and software elements, and with basic PC Linux characteristics.

This publication provides the primary documentation for ISV zPDT and corresponds to zPDT V1 R11, commonly known as GA11.

Authors

Bill Ogden is a retired IBM Senior Technical Staff Member who continues to work part time with projects he enjoys, such as ones for new mainframe users and entry-level systems.

The following people contributed substantially to this publication:

Chris Cook (IBM San Jose) provided team management and key planning and interface functions.

Keith VanBenschoten (IBM Poughkeepsie) now retired and a part-time worker, created zPDT test systems, the “build” process to put together the various parts of zPDT package, and the zPDT installation processes and tools. He managed many issues as they arose.

Theodore Bohizic (IBM Poughkeepsie) is the Senior Technical Staff Member who is responsible for much of the core design and implementation of zPDT.

Frank Kyne (Watson and Walker) provided much-needed assistance with sysplex functions and specialized performance areas.

Alena Yampolskaya (IBM Poughkeepsie) was a member of the zPDT development team. They were helpful with new zPDT functions.

George Darling (IBM Poughkeepsie) contributed updates and understanding of I/O elements and specific IBM zSystems architectures.

Stan King and his team at **Information Technology Company LLC (ITC)** helped with detailed customer operations.

Sandhya Venugopala (IBM India) was a strong developer for internal zPDT functions.

Ragavan Devanathan (IBM India) was involved with specialized zPDT functions and emulated I/O development.

Rajeev Surapareddy (IBM India) was involved with detailed interface projects.

Randy Zurro (IBM Poughkeepsie) managed Linux levels and security requirements for the sizable PC development array that is used for zPDT.

Kevin Dean (IBM Poughkeepsie), **Matthew Portnoy** (IBM US), and **Matthew Sabins** (IBM US) joined the team that developed this zPDT release.

James Guan (IBM Canada) helped the team with detailed performance studies.

John Lee (IBM US) helped the team with newer z/Architecture analysis.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a publication in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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 publications to be as helpful as possible. Send us your comments about this publication 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 email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, IBM Redbooks
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- 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

This chapter provides a brief and somewhat generalized high-level overview of Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT). This chapter is intended for potential or new ISV zPDT customers. *Experienced ISV zPDT users may skip this chapter*.

zPDT is part of several licensed IBM products that use a personal computer (PC) running under Linux to emulate an IBM zSystems mainframe that is used for software development and testing purposes. The emulated mainframe environment uses a licensed IBM zSystems operating system, such as z/OS, IBM z/VM®, or an Linux on IBM zSystems system.[1] The IBM zSystems operating system (with more software) is used to develop and test whatever applications are relevant.

Over the years, the name of the IBM mainframe family has changed several times. Some recent names have been IBM System z®, IBM z System, IBM Z, and others. The current name is IBM zSystems, and this name is used in this publication.

[1] zPDT is also used with IBM z/VSE®, although this scenario is not formally tested by the zPDT developers.

1.1 Frequently asked questions

This section describes some of the frequently asked questions about ISV zPDT.

What is the full name of the zPDT package

At the time of writing, there are two basic zPDT product packages that are intended for development and testing:

1. *ISV zPDT*, which was formerly known as zPDT. The full name is Independent Software Vendor IBM Z Program Development Tool, although this expanded name is rarely used; within this publication, we use ISV zPDT as the short name.[2] This product is intended for a particular set of independent software development companies,[3] and it is also widely used within IBM.
2. IBM ZD&T, which was earlier known as RDzUT and RD&T and zD&T. The full name is IBM Z Development and Test Environment; within this publication, we use IBM ZD&T as the short name. This product is intended for normal IBM zSystems customers, and it involves many licensing and pricing agreements that are established when the product is acquired by a customer.

Much older documentation used the zPDT name as a generalization or because it was written before IBM ZD&T was created. Such documentation usually applies to what is known as ISV zPDT, and it might not include changes that are relevant to the IBM ZD&T product. More recent documentation (including this IBM Redbooks publication) tends to use either the ISV zPDT or IBM ZD&T names.

Both products use the *core zPDT* program[4] as the base element, and the current naming conventions are a bit confusing because this core zPDT program is part of both ISV zPDT and IBM ZD&T. This publication sometimes uses the simple name zPDT when referring to core (or base) characteristics of the main program.

Tip: This publication is primarily about ISV zPDT. While many of the technical details in the publication also apply to IBM ZD&T, you must consult IBM ZD&T representatives or documentation to understand usage differences or extensions that apply to IBM ZD&T.

Some aspects of zPDT technology are present in other IBM offerings, which are not covered in this publication.

What is the purpose of zPDT

ISV zPDT is licensed for software development and some basic software testing. It is not licensed or appropriate as a production system. Defining a software development group within a large organization where there is overlap between "development" and "production" can be confusing, which is a topic for discussion with an IBM representative or a recognized zPDT vendor.

The key point is that ISV zPDT is not valid for application "production" or performance measurement usage. Such usage would violate the licensing agreements for the zPDT product, the IBM zSystems operating system, and other IBM licensed products, and probably other licensed software that is used on IBM zSystems.

[2] The name is often rendered as "Product Development Tool" instead of "Program Development Tool." The extended full name is seldom used and either version conveys the concept of the actual tool.

[3] The qualification to be included in this limited set of software developers is complex and must be reviewed with the appropriate IBM representatives.

[4] The terms *core zPDT* and *base zPDT* often have the same meaning.

Usage of IBM ZD&T can be in a wider scope depending on sales and contractual details. An appropriate IBM ZD&T representative should be consulted for details.

Does either zPDT version include an IBM zSystems operating system

This question involves distinguishing between the core zPDT material and what is available in a product package from IBM:

- ISV zPDT itself does not include any IBM zSystems software or operating systems. However, the license to use ISV zPDT usually contains access to a specially generated z/OS operating system that contains many IBM z/OS software products. The associated z/OS (or z/VM) sets are commonly known as Application Development Controlled Distribution (ADCD) z/OS or ADCD z/VM.
- The IBM ZD&T product is typically licensed to include access to a special z/OS system and (in some cases) to a special z/VM system. Other license aspects are possible.

Viewed a different way, the core zPDT development group within IBM is not directly involved with helping customers with complex IBM software products such as z/OS, large z/OS applications, and others. This basic status applies to ISV zPDT users. The IBM ZD&T product can provide more support in such areas.

Does this IBM Redbooks publication cover both ISV zPDT and IBM ZD&T

In general, the answer is no. This publication applies primarily to ISV zPDT, with only a few references to IBM ZD&T, although this chapter contains some comments about IBM ZD&T and general zPDT information. The remainder of the publication (except for Appendix D, "IBM Z Development and Test Environment notes" on page 387) usually does not mention IBM ZD&T, and normally contains references to ISV zPDT.

Are ISV zPDT and IBM ZD&T priced the same way

No. The licensing and pricing details are quite different and must be discussed with an IBM representative or zPDT vendor.

Can I run either ISV zPDT or IBM ZD&T after I obtain zPDT

ISV zPDT and IBM ZD&T are separate products and require different license tokens. An IBM 1090 token that is used for ISV zPDT does not work with IBM ZD&T, and an IBM 1091 token that is used for IBM ZD&T does not work with ISV zPDT. It is not practical to try to alternate between ISV zPDT and IBM ZD&T. Select whichever one is most practical for you and stay with that one.

How are the two zPDT products derived

Figure 1-1 provides a generalized overview.

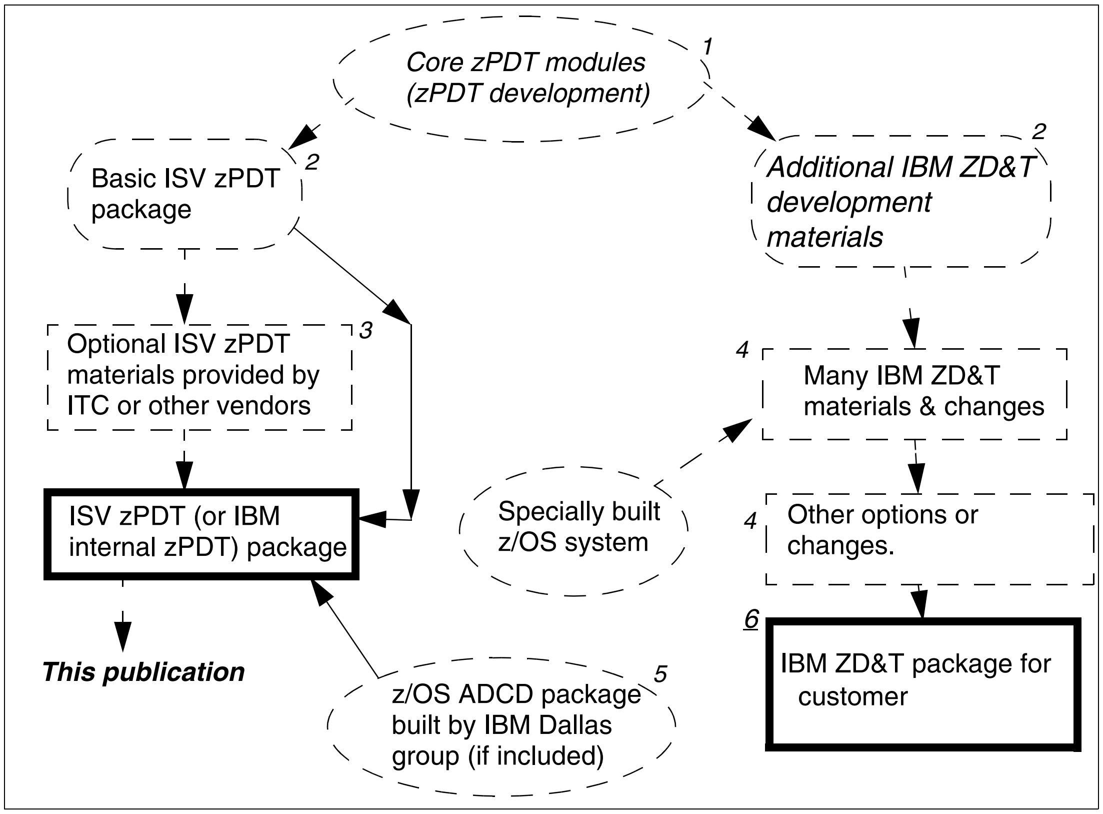

Core zPDT modules (zPDT development)
1
Basic ISV zPDT package
2
Additional IBM ZD&T development materials
2
Optional ISV zPDT materials provided by ITC or other vendors
3
4
Many IBM ZD&T materials & changes
ISV zPDT (or IBM internal zPDT) package
Specially built z/OS system
4
Other options or changes.
6
This publication
5
z/OS ADCD package built by IBM Dallas group (if included)
IBM ZD&T package for customer

Figure 1-1 zPDT overview

The core zPDT modules (1 in Figure 1-1) are standard, and many are referenced throughout this publication. These modules form the basis (2) for both ISV zPDT and IBM ZD&T. For ISV zPDT, it is possible for special vendors (such as Information Technology Company (ITC)) to add operational enhancements that do not interfere with normal ISV zPDT usage (3). IBM ZD&T adds substantial changes and enhancements that can change the basic usage of zPDT (4). The ADCD operating system package is built by a different group in an IBM organization (5). The z/OS version that is potentially delivered to IBM ZD&T customers contains modifications that are not present in the ADCD package that is provided for ISV zPDT users, and might not be associated with the ADCD terminology. IBM ZD&T users (6) might each have different components or different license agreements.

What are the components of the core zPDT modules

In addition to the IBM zSystems emulation module, the core zPDT material contains several Linux commands for zPDT administration and several IBM zSystems I/O device emulation modules. The basic zPDT "component" consists of many lower-level executable or control-level Linux files. At this lower level, an ISV zPDT installation results in three Linux directories:

1. `/usr/z1090/bin` contains about 200 Linux files, which mostly are executable files that are related to ISV zPDT run time.
2. `/usr/z1090/man/man1` contains about 100 brief documentation files for various ISV zPDT commands and operations.
3. `/usr/z1090/uim`, which might not exist until ISV zPDT is started, contains a few, small license-related files.

In addition, there are small files (mostly log files) in a home subdirectory of the Linux user ID that are used to run zPDT.

What are the zPDT tokens that are needed

These tokens usually are physical USB devices that appear the same as small USB flash drives. Although there are exceptions, the tokens usually contain a 1-year license for a number of emulated IBM zSystems processors.[5] The licenses are typically renewed by an online process to update the token. (A more advanced system for IBM ZD&T can involve "software only" licenses.) The tokens have IBM part numbers 1090 (for ISV zPDT) or 1091 (for
IBM ZD&T). The tokens are provided through the same vendor (IBM or a third party) that markets either zPDT product. The USB token (or "software license") must be accessed by the PC running zPDT in order for zPDT to function.

The tokens that are used at the time of writing are so-called "Generation 1" tokens. At some point, there might be a move to "Generation 2" tokens. For practical purposes, Generation 2 tokens are approximately the same as the older version, and the customer usage is about the same. The newer tokens have different program interfaces, but they are internal to the zPDT programs.

You talk about PC Linux, but can I run zPDT under Microsoft Windows instead

zPDT is a complex application that is written to run on a PC Linux base machine. There is no version that is written for Microsoft Windows. All the discussions (in this publication or in various internet forums) assume an understanding of the base Linux requirement.[6]

Is zPDT useful for such things as IBM Parallel Sysplex and multithreading

Yes, but with practical limitations based on PC performance, memory sizes that are available, and others. IBM ZD&T requires a modified license for IBM Parallel Sysplex® emulation.[7] ISV zPDT licenses require no modifications. Parallel Sysplex operation, at the time of writing, is limited to multiple z/OS guest images within a single z/VM image on a single zPDT base system. For more information, see *zPDT Sysplex Extensions - 2020*, SG24-8386.

In general, the same "multithreading" techniques and applications that run on an IBM zSystems mainframe can be used under zPDT. Practical differences, such as memory sizes and emulation performance, might be relevant.

[5] Throughout this publication, *zPDT processors* mean emulated IBM zSystems processing units, also known as Central Processor Units (CPs or CPUs). For this publication, it does not mean a physical "chip" that might contain multiple "cores."

[6] There are ways to run Linux subsets within a Microsoft Windows base, but they are not tested for zPDT, and there is no organized zPDT support for this environment.

[7] This approach involves a special license in the token to enable z/VM to provide the Coupling Facility (CF) function.

What kind of PC is required

The PC can range from a small laptop to the largest PC server that is available. A small ISV zPDT plus a z/OS system, probably with a single z/OS user, might be used with a laptop containing 8 GB of memory and two cores and having *at least* 250 GB of available disk space, but this setup would have limitations.[8] A typical small ISV zPDT plus a z/OS system might be based on a PC with at least 16 GB of memory, about 500 GB of available disk space, and four PC cores. This setup can service multiple z/OS users if no large databases or applications are involved.

A larger ISV zPDT system might have much more than 32 or 64 GB of memory, terabytes of disks, and at least 10 PC core processors. One basic requirement is that the PC must have *at least* as many processor cores as there are defined IBM zSystems processors (CPUs) for ISV zPDT.

The actual required size depends on the number of simultaneous users, the IBM zSystems applications that are involved, the size of IBM zSystems databases that are involved, and others. There is no simple formula for deriving the exact PC size that is required, or even if the intended project is practical on a zPDT PC. If you want a large or complex operation, contact an appropriate IBM representative, or an ITC presentation might be appropriate.

Although a basic PC system can be simple to configure and acquire, the more advanced PC servers can be a bit more complex. If your plans require an advanced machine, contact a knowledgeable IBM or ITC advisor.

Who is ITC

Information Technology Company is an IBM associated vendor who supplies most of the ISV zPDT systems that are now in use and some of the IBM ZD&T systems. They offer several options for extra support levels and depending on your skill levels, these options might be important. ITC also offers PC hardware systems that are especially suited for zPDT.

The primary ITC office is at 7389 Lee Highway, Falls Church, VA 22042 (telephone 800-994-9441). They have more offices in Germany and Great Britain. A general method of contacting ITC is by emailing them at `sales@itconline.com`.

Can I move to a new, faster PC with ISV zPDT

Yes, although your ISV zPDT vendor might want updated information about which PC you are using. The ISV zPDT token itself is not sensitive to a particular PC. However, after you have a general high-end PC, then advancing to a new machine is unlikely to yield much improvement in basic zPDT performance. Areas that often help performance are more PC memory and the replacement of “real” PC disks with solid-state drives (SSDs).

Is zPDT as fast as an IBM zSystems machine, and how fast is it

The simple answer is *no*. Furthermore, there is no linear relationship between zPDT performance and an IBM zSystems machine. Some emulated functions (such as a reasonable sequence of common IBM zSystems system instructions) can be acceptable, and others can be much slower. Although IBM does not provide any detailed zPDT performance specifications, some customers describe performance that is about .06% the speed of a “real” IBM zSystems CPU processor, but with many more comments about various positive and negative exceptions. Such comments and declarations involve considerable background information about the hardware that is involved, the nature of the applications, and other concurrent operations. Again, specific information about performance is not provided by IBM.

[8] The amount of PC disk space that is required might vary over a wide range depending, for example, on the exact nature of your z/OS disk layout (if z/OS is used) and on how much space that you need for your own emulated 3390 volumes.

Some common tasks, such as COBOL compilations, seem to perform well, while others, such as Java usage, tend to be on the slower side. Again, if you have specific goals that are a bit unusual, contact your zPDT source.

It should be understood that the primary goal of zPDT is the accurate emulation of IBM zSystems system instructions, which is an important factor for software developers and takes precedence over performance factors.

How reliable is the ISV zPDT or ZD&T product

This answer can be more complex than expected. We have some ISV zPDT customers who experience months of continuously running their system, but a few complain about weekly problems. Investigations of various situations almost always detect a larger-than-expected variety of system involvements, such as extra Linux root-level monitors (especially internet monitors), unreasonable system expectations, poor (or uninformed) memory planning, odd applications, more (and often poorly defined) Linux loads, and others.

Another factor is the basic reliability of PC hardware, including disk drives, local area network (LAN) connections, LAN administration, power supplies, routers, and others. (This statement is not intended to degrade PC hardware or environments, but it is something that must be considered.)

Are all IBM zSystems functions emulated

No.

Can any application be developed or tested with ISV zPDT

The typical answer is *most but not all*. For example, there can be practical limitations on memory usage (consider an application that randomly and extensively addresses 900 GB of data in memory, running under zPDT on a laptop with 16 GB of memory) and limitations on some special styles of applications (consider an application that wants to do low-level manipulations of an IBM zSystems system cryptographic adapter). If you (as a potential ISV zPDT customer) are considering zPDT usage for a really “odd” or “huge” project, then you might want a specific and detailed conversation with your IBM representative or zPDT vendor.

zPDT systems typically run IBM software such as operating systems, compilers, and others, but this software usually is not seen as “applications.” Applications are typically customer-written programs for whatever data processing development that the customer needs,[9] and zPDT can provide a working tool for developing and testing many such applications. The ADCD z/OS system that is often associated with ISV zPDT has IBM CICS®, IBM Db2®, IBM Information Management System (IMS), IBM WebSphere® Application Server, and IBM z/OSMF preconfigured for basic use, and they can be “starting points” for some application development work.

We have some old PC tape drives: Can they be used by z/OS

The answer is *possibly*. During the years of typical PC availability, there has been a wide range of “real” tape drive options, some involving detailed selections of cables types, cable speeds, PC adapters, and adapter options. The current ISV zPDT versions are sometimes briefly tested with a limited set of these options, and the ISV zPDT developers are aware that there are relatively few ISV zPDT tape drive users. Contact ITC about tape drive questions. Specific information about the relevant tape drives (and adapters and cables and control options) is needed.

[9] Customers using only Linux for IBM zSystems might see this IBM software topic differently.

Also, ISV zPDT can be configured with *emulated* tape drive operation, where the emulated tape is just a Linux disk file. This usage is much more widely used than “real” tape drives.

My z/OS skills are limited: Does ISV zPDT provide simple assists for z/OS

No. ISV zPDT emulates an IBM zSystems hardware machine. If you run z/OS on an ISV zPDT system, then you (or someone helping you) need some basic z/OS skills. Roughly the same degree of skills are needed whether running z/OS on zPDT or on a “real” IBM zSystems machine.[10] The z/OS ADCD system that is typically associated with an ISV zPDT license is a “real” z/OS system in that it was configured and tested on a “real” IBM zSystems machine. z/OS (or any other IBM zSystems operating system) is not part of the core zPDT program.

Is ISV zPDT or IBM ZD&T suitable for basic z/OS learning

The “underlying” z/OS version is the key point. With ISV zPDT (and “standard” IBM ZD&T), the underlying z/OS is assumed to be a fairly complex z/OS environment. The typical customers often have many years of IBM zSystems and z/OS or z/VM experience,[11] and the ADCD z/OS systems that are provided with the ISV zPDT product contain many complex options and setups. Also, the documentation that is involved (such as this publication) is not written as basic IBM zSystems learning material.

zPDT that is combined with a more basic and simplified z/OS system *can* be used for practical IBM zSystems learning, although this learning benefits from different options for the z/OS configuration, a different set of practical educational documentation, access to relevant help for the student, and a specialized license agreement. The zPDT program itself is the same in all cases.

Does IBM provide an IBM zSystems Linux package for ISV zPDT

No. You must find and download a distribution of Linux that is built for use on IBM zSystems machines. Many of our ISV zPDT customers have done this task, and several versions or levels of Linux for IBM zSystems machines exist. These distributions of Linux are not IBM products, and the ISV zPDT development team does not attempt to test every one of these distributions. Over the years, some problems were detected, mostly related to attempts to install Linux for IBM zSystems from DVD if the new image contains some obscure change to the DVD format or the installation details.

New IBM zSystems Linux versions sometimes contain low-level alterations that do not routinely work with the zPDT emulation available then, and they might depend on clever bypasses or future zPDT releases.

Obtaining more IBM software for ISV zPDT

“I want to order ISV zPDT and the z/OS ADCD system. Then, I want to order more IBM software that is not already included in the z/OS ADCD system. How do I do this task? How do I acquire IBM software fixes?”

[10] Some basic skills differ, such as controlling a system start and the initial start of an operating system. These functions are much simpler on a basic zPDT system.
[11] zPDT is sometimes used for z/VSE. For more information, check with your z/VSE provider.

In a simple environment, none of these items are routinely available. ISV zPDT and the ADCD systems do not provide a general license for more IBM software or software maintenance. The ADCD systems that are used with ISV zPDT are special cases of unique IBM license handling. In the most general case, ordering more IBM software requires more licenses for that software, and the licenses that are involved probably are more expensive and less relevant for ISV zPDT usage.

It is important to understand the IBM license limitations for ISV zPDT and ADCD systems. These offerings bypass the normal license fees that are associated with IBM zSystems machines and related IBM software. However, this “bypass” also bypasses access to normal IBM Support, maintenance (such as software fixes), and the ability to request IBM software changes. New ADCD releases for z/OS are generally offered every 6 months, with some variations in the timing. Assuming that your ISV zPDT license agreement includes access to ADCD systems, this agreement is the primary “IBM software maintenance” that is offered for ISV zPDT customers. However, the ADCD implementation group does offer a method of downloading IBM maintenance service (program temporary fixes (PTFs)) and some APAR information) in the form of monthly available downloadable large files. Substantial z/OS experience (and considerable time) is needed to handle this approach, and few ISV zPDT users attempt it.

I want to use my company’s installed z/OS instead of the ADCD version

In principle, the normal ISV zPDT license includes access to the current ADCD z/OS version and no other version of z/OS. You should discuss this topic with your IBM representatives for a possible variation to the normal ISV zPDT or IBM ZD&T license terms. An important consideration is the practical needs of your “company” z/OS. Do you expect cryptographic adapter adjustments, or Parallel Access Volumes (PAVs), or HiperSockets, or any of the more complex disk control unit functions? A reasonably skilled z/OS administrator[12] can manage such issues for ISV zPDT, but you must have such an administrator available.

I want z/VSE software development: Is there another zPDT publication for me

No. This publication mostly describes ISV zPDT usage for a z/OS environment. ISV zPDT often is in a z/VSE environment, although there is no formal IBM testing involved. Many of the ISV zPDT details in this publication should also apply to z/VSE environments.

How can I protect my zPDT license token

An ISV zPDT token is required in a PC USB port to use ISV zPDT. (A special exception exists for IBM ZD&T “software licenses.”) The token can easily be unplugged and removed, although this action stops any ISV zPDT operation. One security option for some high-end PC servers is a hardware design for installing USB “sticks” internally in the PC. Another potential security option is to install the zPDT token on a remote zPDT license server where this remote server is in a more secure location. (ITC offers a special small remote device for this approach.)

[12] The older term was “systems programmer” or “sysprog”, but “administrator” is the current terminology.

Which PC Linux should I use for my base

Several Linux distributions are mentioned in this publication. However, we stress that the latest Linux releases, or the latest updates or fixes, are sometimes not the best choices. ISV zPDT is a rather complex Linux application, and the latest Linux changes sometimes create hidden issues.[13] ISV zPDT usage is typically a major undertaking, and the PC Linux that is involved is generally dedicated to this purpose. A stable, reliable Linux base, showing good ISV zPDT operation, is often regarded as better than the latest Linux alterations.

Tip: ISV zPDT installation is usually easier if you use one of the Linux distributions and levels that are listed in this publication. (See 2.5.1, "Current release" on page 44 for details.) These bases are the ones that were used for testing the current ISV zPDT release. Attempting to use different Linux distributions or levels might lead various minor issues.

What about new Linux releases and fixes: Are they good for zPDT

Maybe. zPDT installation is mostly handled by an installation program that is part of the zPDT package. Linux releases and updates often have minor internal changes that are relevant to zPDT and sometimes cause the zPDT installation process to encounter problems. There are usually fixes or bypasses that are needed to overcome such problems, but using them creates a delay that might be inconvenient. New ISV zPDT releases are typically based on the Linux levels current at the time that the release development started. One solution to this question might be to stick with the listed Linux levels, when possible, and update them when installing a new ISV zPDT release.

Some Linux updates (assuming you decide to apply them) require a Linux restart and perhaps some minimal testing. Depending on your usage of zPDT operation, this approach might require more planning on your part.

How do I access z/OS or z/VM on my ISV zPDT machine

You can install ISV zPDT and the appropriate IBM zSystems operating system on one PC (with a base Linux) and use 3270 emulators on the same PC (running under the base Linux) or on other machines (whether they are Linux systems). The 3270 emulator can be x3270, or almost any other modern 3270 emulator product. Accessing your zPDT setup from other PCs requires network connections, and there can be many variations of this operation.

How much PC Linux skill do I need to use ISV zPDT

You need basic Linux skills to install and administer ISV zPDT, but no in-depth Linux development work is involved. ISV zPDT, in general, does not depend on specific Linux desktop options or the existence of a Linux GUI, but it might (depending on your usage plans) need Linux firewall management and communications setups. These items are largely Linux options, not ISV zPDT options, and they might be important to your Linux or ISV zPDT users. Someone in your organization might need a skill or outside assistance to produce a Linux installation that is acceptable for your users.

An important area is LAN usage. zPDT LAN usage (through IBM zSystems Open Systems Adapter (OSA) emulation) works well with basic LAN interfacing. Dealing with VPNs, firewalls, DHCP, multiple OSA adapters (especially when connected to the same LAN network), complex or changing virtual or container usage, router changes, and others can require more networking skills.

[13] These issues sometimes include Linux library level changes, library directory changes, LAN and firewall administrative changes, and others. Such changes can require corresponding zPDT administrative changes that might not be immediately obvious.

We have seen a few cases where the Linux owner has “adjusted”, “customized”, “built from scratch”, or “experimented with details that they did not understand” and encountered ISV zPDT problems. A clean installation of one of the Linux distribution levels that are listed in Table 2-3 on page 45 should not encounter unusual problems.

Also, recent Linux distributions that are used for zPDT are tending to be distributed as “large” ISO files that do not fit on a DVD, even though they might have “DVD” in their release title. In practice, this approach usually means that the distribution must be downloaded to a USB flash drive that, in basic usage, contains nothing but the ISO files.

Can I use a virtual or container base

Some ISV zPDT customers use virtual or container bases, which are acceptable, although intelligent planning is needed. For example, these environments often over-commit PC resources, or create “containers” that are much too small for reasonable zPDT performance, need more complex communications arrangements, or present reduced zPDT administrative interfaces. As a best practice, first use an ISV zPDT in a simple, non-virtual, non-container environment to establish basic familiarity and operating skills. Linux memory management, especially in virtual or container environment, can be a more complex topic than many desktop users appreciate.

Must my IBM zSystems applications be modified to test them on an ISV zPDT system

In general, the answer is *no*. In fact, rather specialized low-level programming is needed for a program to detect that it is running under zPDT emulation. However, this answer assumes that your applications reasonably fit the speed, the memory size, and the available disk space on the PC platform being used, and that you can handle whatever internet connections are needed.

I need multiple IBM zSystems disks for my own programs and data: Is this scenario possible

Yes. The ADCD z/OS system has some available space on the various operating system volumes, but you can create many more “work” or “data” volumes. Assuming that you have sufficient PC disk space and memory and that the workload is reasonable on your ISV zPDT system, you can create dozens (or hundreds) of additional emulated IBM 3390 volumes. Each volume can have (3390 style) 1 - 65 K cylinders for normal volumes, or be much larger for extended address volume (EAV) volumes.[14] How many emulated IBM 3390 disk volumes might be practical? The answer depends on PC disk space and a detailed understanding of your workloads. There is no standard formula.

ISV zPDT provides two methods for copying 3390 volumes from a “real” IBM zSystems system. Both methods depend on network connections (or optionally, USB flash drives in one case) and do not involve a “live” ISV zPDT operation.

Can I attach a “real” IBM 3390 volume or control unit to ISV zPDT

The simple answer is *no*. There are no hardware IBM zSystems channel adapters for ISV zPDT.

[14] If “EAV volume” is an unfamiliar term, you might want to read about it on the web. A 1 cylinder 3390 volume would be unusual, but is possible.

How large is the ADCD z/OS system, disks, and memory

Memory size for z/OS (on a “real” IBM zSystems machine or as defined for zPDT) typically starts around 4 GB (or 8 GB for a “real” system), but is usually considerably larger. How large? The answer depends on the workload. A reasonably skilled z/OS administrator, after understanding your workload, can help with guidelines.

At the time of writing, the more basic form of the z/OS ADCD system needed 15 - 28 disk volumes, depending on certain choices.[15] A few more volumes are needed if you want Db2, IBM CICS, IMS, or other specialized options. These volumes are usually IBM 3390-9 volumes (8.5 GB each). In addition, volumes for your programs and data might be needed.

Is my data secure on zPDT and as secure as on a “real” IBM zSystems system

Maybe, but unlikely. You must ensure absolute security for the base PC and the base Linux, manage z/OS running under zPDT (usually through IBM RACF®), and not install the same cryptographic adapter master key that is used on the “real” IBM zSystems system. You also must manage access to the zPDT token license. In principle, all these tasks are possible. In practice, it is unlikely. As a best practice, do not move confidential data to a zPDT system. Again, ISV zPDT is for software development and testing, which typically does not involve confidential production data. Another aspect involves “backups” for your Linux system. Anyone who can access the backups can potentially access any of the emulated IBM zSystems volumes that are part of the backup.

Can ISV zPDT be an expanded system

“ISV zPDT sounds good. I need to concurrently run 5 or 6 copies of z/OS, and a few copies of VSE, and perhaps two CF versions. How much memory do I need? Which PC server?”

This setup might or might not be reasonable for an ISV zPDT solution. IBM zSystems system emulation on a PC (such as ISV zPDT uses) is possibly not a good choice for an environment this large and complex, depending on more detailed information about your software, general environment, and usage patterns. Having stated this warning, some ISV zPDT customers do operate rather complex, large systems. although they are based on appropriate planning and testing. Contact an IBM representative or a zPDT vendor about your specific needs.

How many IBM zSystems CPUs can I create with zPDT: More CPUs allow more work

A single instance of ISV zPDT can have up to eight emulated IBM zSystems CPUs. (Some of these CPUs can be configured as IBM zSystems Integrated Information Processor (zIIP), IBM zSeries Application Assist Processor (zAAP), or IBM Integrated Facility for Linux (IFL) processors, but they count toward the limit of eight. SAPs and ICFs are not possible. Emulated CFs are possible but not counted.) A zPDT license is needed for each CPU (including zAAP and IFL processors, but not zIIP processors), which might involve more than one zPDT token. The base PC must have at least as many processor cores as there are active zPDT CPUs. Multiple instances of zPDT can be used with each instance with the same limit of eight if there are sufficient zPDT licenses. Again, as a best practice, consult a zPDT representative (at IBM or ITC, for example) if you are considering a large zPDT configuration.

Consider the performance of a realistic zPDT system. For example, it is possible to have multiple z/OS systems running under z/VM or as separate zPDT instances (assuming that you have sufficient zPDT licenses). How effective this setup might be depends on the workloads on the multiple z/OS systems.

[15] A much smaller disk environment for z/OS can be practical. The number of volumes that are mentioned here reflects the current ADCD version of z/OS.

How are logical partitions handled by ISV zPDT: How many can I have

ISV zPDT does not provide logical partitions (LPARs), although some “LPAR terminology” is used. Multiple *instances* of ISV zPDT can be used, which provides an effect like LPARs. z/VM can be used with multiple “second-level” operating systems under it, which provides another effect like LPARs.

Specific configuration limits for ISV zPDT: Where can I find such information

“My staff says that they need 19 LAN interfaces, 427 disk drives (emulated IBM 3390 units), and some remote automatic backup or duplicate disk volumes. They say we use PAV for our disks. Is this scenario possible?”

Such generalized specifications are not applicable to ISV zPDT, although there are a few specific limitations in zPDT implementation. For example, there is a maximum of 32 connections for “local” 3270 terminals,[16] and a maximum of 2048 emulated devices, but such configurations are not necessarily practical. ISV zPDT is intended for more basic software development and simple testing, and it is not intended to replicate a larger IBM zSystems system configuration.

Your staff’s example of 19 LAN interfaces might be possible, but it is outside any routine testing that is done during zPDT development. Your 427 emulated disk drives (where you did not specify the size of each 3390 volume) can be practical if there is sufficient memory and disk space on your base PC. “Remote” disk interfaces can be a complex topic with modern IBM zSystems servers, but is generally outside zPDT functions.[17] At the time of writing, ISV zPDT does not emulate PAV. The most important information that is missing from your staff’s note is a *detailed* understanding of the nature of the workloads that they expect to develop and test on ISV zPDT, especially where a complex configuration is involved.

Chapter 2, “ISV IBM Z Program Development Tool concepts and terminology” on page 19 provides a summary list of IBM zSystems system features and functions that are not provided by ISV zPDT.[18] This summary can be useful, but it does not address an important side issue. ISV zPDT customers are assumed to have a reasonable degree of IBM zSystems system skills and the ability to set up whatever reasonable reconfiguration is needed (if any) for their programs or operating systems. However, few ISV zPDT customers have reported serious problems in this area.

Does ISV zPDT allow multiple concurrent users, and how many

Assuming the relevant IBM zSystems operating system that is running on the ISV zPDT system allows multiple concurrent users, the answer is *yes*, that is, multiple users are a normal situation. *How many* is a more difficult question. A typical ISV zPDT might have 1 - 5 (or 1 - 10) concurrent users as an example. (We assume that they are Time Sharing Option (TSO) users, but they might be related to CICS, IMS, or another application.) There is no specific maximum number, but dozens of users, for example, might not be reasonable.

[16] The meaning of “local” 3270 connections (in zPDT, the emulation of such connections) has specific meanings for people familiar with IBM zSystems servers. There is no specific limitation for OSA-connected terminals.

[17] NFS can be used for “remote” disk connections if the response times are good, but it only provides more disk space. RAID functions can be used at the PC hardware level, which is not apparent to zPDT.

[18] Some of the topics in the list can be considered “high level”, and a basic understanding of the topic might be needed to explore practical details.

I understand that z/OS needs 3270 terminals: Does ISV zPDT provide them

ISV zPDT provides interfaces for 3270 emulators, but it does not provide a 3270 emulator. However, most Linux systems (as used with the base PC for ISV zPDT) can obtain the x3270 emulator,[19] which works well. (If the emulator is not included, it can be downloaded from multiple sites.) Some ISV zPDT customers use various other 3270 emulators. With proper networking, the 3270 terminal emulation does not need to be limited to Linux bases.

Are new zPDT releases or updates available and support new IBM zSystems machines

Yes, although the schedule for new releases is variable. In addition, “fix pack” releases are sometimes released. Installing a new zPDT release or fix pack is simple (and easily reversed in the unlikely event that something goes wrong with the new modules). New zPDT tokens are not required when moving to a new zPDT release. Generally, new hardware instructions that are provided by new IBM zSystems mainframes are also provided by new zPDT releases that occur after the new mainframe is first delivered.[20]

Can you provide a “script” for installing the base Linux that is needed for zPDT

No. The various commands and options that are needed for installing Linux vary with different Linux distributions, updates, additional Linux “packaging” that is based on one of the three formal Linux distributions that used by the zPDT developers, and on the specific set of optional functions. Some of these details are relatively minor, but these minor issues can change frequently. However, we believe that there are no special issues that are involved with the Linux base for zPDT that was used for developing that specific zPDT release.

Must my base Linux that is used for ISV zPDT have internet access

There are several answers to this question:

- Internet access is not needed for normal ISV zPDT operation when you use a zPDT token that is installed in the base Linux PC. (This answer assumes that the zPDT owner does not need internet access for specific purposes, such as connections for TSO users.)
- If a remote zPDT license server is used (or a “software” license for ZD&T), then some LAN access is needed, which can be a local LAN or the full internet, depending on various aspects of the configuration.
- During a zPDT installation or upgrade, or a base Linux installation or upgrade, a normal internet connection *might* be needed to obtain the necessary Linux library files. This aspect of zPDT installation can change with each zPDT release or upgrade or with various Linux distributions or upgrades.

How prepared are the ADCD z/OS system and the ADCD z/VM system

For ISV zPDT, both these systems can be installed, started, and used for basic, normal functions. The z/OS system typically includes TSO, IMS, CICS, Db2, IBM Interactive System Productivity Facility (ISPF), System Display and Search Facility (SDSF), TCP/IP functions, and various other IBM software. Neither system has detailed setups for complex, ready-to-use exercises or demonstrations. The user is expected to adjust and configure the system (and the software products) for specialized undertakings.

[19] x3270 is an open source program that is maintained by Mr. Paul Mattes at the time of writing. It is not an IBM product.

[20] Not *all* new instructions might be provided. Excluded instructions are typically specific functions that might be used only by the operating system. Thus far, such limitations have not prevented general usage of the IBM zSystems operating systems.

What IBM “support” goes with ISV PDT: IBM Help and problem resolutions

In a contractual sense, ISV zPDT is “unsupported” both for zPDT itself and for any IBM software (such as the ADCD z/OS system) that is grouped with it. In a practical sense, related zPDT questions are often informally handled in many ways, such as through web groups and forums. There is no direct contractual “support” for ISV zPDT ADCD z/OS questions, so customers are expected to have their own relevant z/OS or z/VM skills. In addition, there are web groups and forums that can be helpful in many areas.

More “support” can be available for ZD&T customers, depending on the specific nature of the ZD&T contractual arrangement. Also, many support options are available through ITC.

Can I maintain earlier programs, z/OS systems, and IBM zSystems machines

At the time of writing, new ISV zPDT distributions only emulate IBM zSystems machines. IBM does not maintain or provide older zPDT versions that emulate older IBM zSystems machines. Likewise, ADCD z/OS distributions are based on the version of z/OS at the time of writing. If you obtain older zPDT versions and older z/OS versions, they might require older base PC Linux versions and older PC hardware. In principle, you can use this approach, but there is no IBM assistance for doing it.[21] Also, zPDT can emulate IBM 3380 and 3390 direct access storage device volumes, but it cannot emulate older direct access storage device volumes.[22]

Can my ISV zPDT system share data with my company’s PCs

zPDT emulates an IBM zSystems machine. Much of the data on an IBM zSystems machine is in Extended Binary Coded Decimal Interchange Code (EBCDIC) format, but it can include multiple forms of floating point data, ASCII, Unicode, packed decimal data, and many binary formats. These formats are not typically used by PCs. There are techniques for communication between an IBM zSystems system (“real” or zPDT) and PCs, but they seldom appear as “universal” sharing of data. You can upload or download files, for example, from one platform to the other, but the usefulness depends on the specific format of the data involved.

What happens if I let the zPDT token expire

A zPDT token contains the licenses that are needed to run zPDT. Each license has an expiration date. If the license expires, then zPDT stops operating. Your PC or your data is not erased, but a zPDT stoppage might affect whatever data is being processed at that specific time. Normal zPDT usage involves renewing token licenses through a relatively simple email or web process.

My staff says I need zPDT, but I do not understand many of the terms here

The text in this introductory chapter uses many terms and abbreviations that should be familiar to users with IBM zSystems system experience, but it might be a bit obscure to others. There are different levels of such experience, for example, COBOL programmers, database developers, systems programmers (or “administrators”), and production operators. The ISV zPDT products are primarily oriented toward users with IBM zSystems background skills. You might want to review this material with your staff members!

[21] A few, long-time ISV zPDT customers use this approach by accumulating, over the years, older zPDT releases, older ADCD z/OS releases, older base Linux versions, and older PC hardware.

[22] IBM 3380 emulation is provided by zPDT, but it is rarely used.

Also, there are more detailed technical questions and answers that are covered in Appendix A, “FAQ” on page 361.

Extended operation

“My staff reminded we that we need our development systems to run for extended periods, perhaps weeks or months without any interruptions or downtime, with remote users having solid access. Is this practical?”

Maybe. The primary target for ISV zPDT design is basic development operations, such as programmers entering programs, compiling them, and then performing unit tests, probably by using 3270 ISPF interfaces. This approach is not quite what your staff is specifying. Also, perhaps some investigation into the “development and test” contractual issues might be appropriate.

Nevertheless, some ISV zPDT customers routinely operate in modes somewhat like what your staff requested. ISV zPDT is a lightly support Linux application package, and it does not offer the solid architectural base of a “real” IBM zSystems system.

Does this IBM Redbooks publication represent a contract for obtaining zPDT

No. This publication can be helpful for understanding and using zPDT products, but it does not represent a contract or provide the contractual agreement that is needed. IBM Redbooks publications, including this one, are generally based on experiences by a group (often including both IBM employees and IBM customers) that use the relevant products.

What is in the rest of this publication

This IBM Redbooks publication is not intended to be read from front to back. Various chapters and topics are intended for reference only when handling specific situations. This publication covers the following topics:

- Chapter 2, “ISV IBM Z Program Development Tool concepts and terminology” on page 19 contains a more detailed description of ISV zPDT characteristics, modules, tokens, PCs, Linux directories, previous ISV zPDT releases, performance elements, and others. It does not contain detailed “hands-on” directions for system usage.
- Chapter 3, “Device maps for ISV IBM Z Program Development Tool” on page 49 describes the device maps (devmaps) that are used to configure an ISV zPDT operational environment. This information is critical for creating an operational ISV zPDT environment.
- Chapter 4, “ISV IBM Z Program Development Tool commands” on page 71 describes ISV zPDT commands, that is, commands that are issued through the PC base Linux to configure, control, or manage the ISV zPDT operation. There are many of these commands, although the typical ISV zPDT operation involves only a few of them.
- Chapter 5, “ISV IBM Z Program Development Tool installation” on page 125 describes the specific steps that are needed to install ISV zPDT.
- Chapter 6, “Application Development Controlled Distribution installation” on page 137 provides a brief overview about installing an ADCD z/OS system and the procedures for starting and stopping it.
- Chapter 7, “Local area networks” on page 149 provides information about configuring LAN connections for ISV zPDT and for direct LAN connections to an ADCD z/OS. The total view of several techniques can be a bit confusing, and this chapter might provide difficult reading for some users.

- Chapter 8, “ISV IBM Z Program Development Tool licenses” on page 179 describes handling of ISV zPDT licenses (from tokens), and includes extended options for remote token or license operation. Multiple token (license) options also are described. Many basic ISV zPDT users are probably not involved with these details.
- Chapter 9, “Other IBM zSystems operating systems” on page 201 describes IBM zSystems operating systems other than z/OS, mostly involving IBM z/VM. A few practical details about z/VM usage are included.
- Chapter 10, “Multiple instances and guests” on page 215 provides a brief description about multiple ISV zPDT instances. This topic is a relatively complex one that is of concern for a small set of ISV zPDT customers.
- Chapter 11, “The awscmd device manager” on page 227 describes the awscmd function, which is a pseudo-emulated tape drive that can send simplified commands from within ISV zPDT to the base Linux.
- Chapter 12, “Minor z/OS notes” on page 237 contains notes about minor z/OS options or techniques that are referenced by typical ISV zPDT users.
- Chapter 13, “Additional ISV IBM Z Program Development Tool notes” on page 273 contains minor ISV zPDT notes about specialized topics.
- Chapter 14, “Tape drives and tapes” on page 303 contains limited material about using PC hardware tape drives with zPDT.
- Chapter 15, “Direct access storage device volume migration” on page 313 contains information about moving (“migrating”) disk contents from a “real” IBM zSystems to emulated IBM zSystems volumes on ISV zPDT.
- Chapter 16, “Channel-to-channel” on page 323 describes emulated channel-to-channel (CTC) operations between two ISV zPDT systems.
- Chapter 17, “Cryptographic usage” on page 331 provides some basic cryptographic function usage. This specific information might change with new ADCD releases, and the reader is assumed to have some IBM zSystems cryptographic skills.
- Chapter 19, “Problem handling” on page 355 contains a brief description about ISV zPDT problem reporting and handling.
- Appendix A, “FAQ” on page 361 contains many brief question and answers that frequently arise in ISV zPDT classes and reviews.
- Appendix B, “Non-QDIO Open Systems Adapter” on page 375 describes non-QDIO network usage.
- Appendix C, “Generation 2 tokens and licenses” on page 379 describes Generation 2 tokens for ISV zPDT and IBM ZD&T.
- Appendix D, “IBM Z Development and Test Environment notes” on page 387 describes some differences between ISV zPDT and IBM ZD&T.
- Appendix E, “Secure x3270 connection” on page 391 describes a setup for encrypted usage of the x3270 terminal emulator.

2

ISV IBM Z Program Development Tool concepts and terminology

Important: This IBM Redbooks publication contains material from Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) GA11. Some of the details might not match earlier versions of ISV zPDT.

This chapter contains various topics that provide conceptual information about ISV zPDT technology. Later chapters provide specific installation, operation, and management details. The topics in this chapter are grouped as follows:

- Some of the terminology that is used throughout this publication.
- 2.2, “IBM zSystems characteristics for ISV zPDT” on page 21 describes higher-level design details for ISV zPDT, and it also describes the practical usage and limitations of the design. Topics include the Extended Binary Coded Decimal Interchange Code (EBCDIC) character set; emulated I/O devices; excluded IBM zSystems features; ISV zPDT design and operational environment considerations; security; reliability, availability, and serviceability (RAS) considerations; ISV zPDT performance; concurrent personal computer (PC) workloads; IBM zSystems architecture levels; and virtual environments.
- 2.2.9, “Hardware tokens” on page 33 provides a brief description of token details.
- 2.3, “PC selection overview for ISV zPDT” on page 35 provides a description of the practical details that are involved when you select or configure a PC for ISV zPDT usage. Topics include a general overview of PCs to be considered, some basic PC notes, PC memory use understanding, PC disk space for emulated direct access storage device volumes, and an “official” IBM statement about PC selections for ISV zPDT.
- 2.4, “Practical ISV zPDT operational notes” on page 38 describes Linux software levels, Linux user IDs, ISV zPDT operational components, consoles, devmaps, Linux directories, PC local area network (LAN) topics, and multiple ISV zPDT instances.
- 2.5, “ISV zPDT releases” on page 44 contains notes about the ISV zPDT release (at the time of writing) and minor notes about previous releases.

As a reminder, this publication is about the ISV zPDT product. Many of the details that are provided also apply to the IBM IBM Z Development and Test Environment (ZD&T) product, but you must consult IBM ZD&T documentation for more information. A brief overview of IBM ZD&T differences is in Appendix D, "IBM Z Development and Test Environment notes" on page 387.

2.1 Terminology

In this publication, we use the following terminology:

- The *base machine, underlying host, underlying Linux,* or *host Linux* is the Intel compatible PC that is running Linux.
- *z/OS* is used to refer to a recent release of the z/OS operating system, and likewise for z/VM and others.
- A *device map* (*devmap*) is used to specify the operational configuration of ISV zPDT. It is a simple Linux flat file.
- A *token* or *license key* refers to a hardware device that supplies an ISV zPDT license. The terms token, key, and license are used interchangeably. A license also can be supplied by a software-only mechanism when using IBM ZD&T. One license is needed for each IBM zSystems CP emulation. *CP* refers to a general IBM zSystems processor that is the major functional element of ISV zPDT. By default, ISV zPDT provides IBM zSystems CPs. Optionally, you can convert a CP to a IBM zSystems Integrated Information Processor (zIIP), IBM zSeries Application Assist Processor (zAAP), or IBM Integrated Facility for Linux (IFL) processor.[1]
- *Processor* or *core* normally refers to the Intel or AMD processor cores in the base machine. A 2-core machine has two processors, although both are typically in one hardware "processor" module.
- *Open Systems Adapter (OSA)* is sometimes used as shorthand for an *OSA-Express adapter*. The ISV zPDT system provides OSA-Express6s emulation at the time of writing.
- Many Linux commands for the base Linux system, are shown throughout this publication. If the command is preceded with # (a hash or pound symbol), the command is entered in root mode. If the command is preceded with a dollar sign ($), it is not entered in root mode. The mode is important: Do *not* attempt to use zPDT constantly as root.
- ISV zPDT releases are denoted by GA6, GA7, GA8, and so on, where GA means *general availability*. GA10, for example, means ISV zPDT Version 1 Release 10. All ISV zPDT releases, to date, have been Version 1. GA10.1 means the first "fix pack" for the GA10 release.
- IBM zSystems I/O devices have *device numbers* that are used to specify a specific device or interface. An older term for a device number is *address*, and this older term is still widely used. This publication uses address and device number interchangeably.
- Just In Time (JIT) technology has different meanings in different environments. In ISV zPDT, it provides a method of improving performance by consolidating the emulation of a string of IBM zSystems instructions into an optimized string of Intel-compatible instructions.

[1] Using more general IBM zSystems terminology, ISV zPDT provides processing units (PUs). By default, the PUs are characterized as CPs, but can be characterized as zIIP, zAAP, or IFL processors instead. Throughout this publication, we refer only to CPs, and this reference should be understood to include zIIP, zAAP, and IFL processors when they are used. zAAPs are no longer available at the IBM z14® level, but were available in earlier ISV zPDT releases.

The primary operational characteristic of ISV zPDT, in which the instruction set of one computer platform (IBM zSystems) 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 others. We attempt to avoid all this terminology and simply refer to the ISV zPDT product.

This publication supports the ISV zPDT product. Many of the details also apply to the IBM ZD&T product, but any differences are not described here. Appendix D, “IBM Z Development and Test Environment notes” on page 387 describes some of the minor differences as related to tokens.

2.2 IBM zSystems characteristics for ISV zPDT

ISV zPDT functions include IBM zSystems processor (CP) operation and the emulation of various I/O devices. As a general statement at the time of writing, all the functions (instructions and I/O) that are needed to run IBM zSystems operating systems are provided.

2.2.1 ISV zPDT character sets

ISV zPDT character data is typically in EBCDIC, which is true for any IBM zSystems 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 IBM zSystems operating system and applications. There is no routine translation to the ASCII of the underlying host Linux system. The same binary data representation that is used on large IBM zSystems servers also is used on ISV zPDT systems. This representation extends to fixed point, packed decimal, and all floating point formats. All ISV zPDT data is in IBM zSystems representation.

IBM zSystems software running in an ISV zPDT environment is binary compatible with large IBM zSystems machines. For example, application programs that are compiled and linked in one environment can generally run unchanged in the other environment,[2] assuming that the configuration elements are compatible. There can be a few exceptions to this compatibility, usually based on rather obscure application designs.

There are special cases 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. (Usage of zPDT emulated card readers and printers is rare.)

[2] This general statement assumes that relevant operating system and other libraries are at compatible levels. It also assumes that relevant licenses allow such use.

2.2.2 ISV zPDT emulated I/O devices

An ISV zPDT system includes 12 *device managers*, each of which provides emulation for a related group of devices. A device manager can emulate multiple instances of its devices.

aws3274	Emulates a local, channel-attached 3274 control unit. This device manager is almost always used to provide the IBM MVS operator console and 3270 application sessions. Each terminal appears (to the IBM zSystems operating system) as operating through a channel-attached non-Systems Network Architecture (SNA) DFT IBM 3274 control unit. TN3270 sessions are used through the base Linux TCP/IP interface. Typically, 3279 terminals (and rarely, 3284 printers) are the devices that are emulated.
awsckd	Emulates IBM 3390 (and IBM 3380) disk units by using a single Linux file for each 3390 or 3380 device. 3390-1, -2, -3, and -9 are the most common devices that are emulated, although 3390 volumes of almost any size can be emulated.
awsosa	Emulates an IBM OSA-Express6s adapter in either QDIO (OSA-Express Direct (OSD)) or non-QDIO (OSA-Express (OSE)) mode. The hardware that involved is an Ethernet adapter on the underlying PC.[3] This device manager can support TCP/IP operation. SNA operation is not supported.[4] It can also support Open Systems Adapter/Support Facility (OSA/SF) usage when using older IBM zSystems operating systems that provide it.
awstape	Emulates a 3420, 3480, 3490, or 3590 tape drive by using a Linux file in place of the tape media.
awscmd	Emulates a tape drive, but routes output records to the base Linux system, where they are run as commands, and returns Linux output to the emulated tape drive.
awsfba	Emulates Fixed Block Architecture (FBA) devices, which are supported by z/VM and a few other operating systems. A Linux file is used for each emulated device. This device manager is *not* the Fibre Channel (Open Systems) FBA on recent IBM zSystems machines. IBM 3996-1 or -2 devices are emulated.
awsoma	Emulates the Optical Media Attach interface, working with Linux files in this format. This function is read-only.
aws3215	Emulates a 3215 console device (seldom used today) by using a Linux terminal window for the interface.
awsprt	Emulates a 1403 or 3211 printer by using a Linux file for output. Provides emulation of a 1403-N1 or 3211 printer. Forms Control Buffer (FCB) emulation for 3211 is provided, but Universal Character Set (UCS) functions are not provided. Automatic ASCII translation (fixed translation table) is provided.
awsrdr	Emulates a 2540 card reader by using Linux files as input. (The 2540 card punch functions are not emulated.) Both EBCDIC and ASCII data can be used.

[3] Wireless can be considered an Ethernet adapter.

[4] Initiating SNA operations (in non-QDIO mode) might be possible, but this usage has not been tested and is not supported by IBM at the time of writing.

awsscsi	Uses a Linux SCSI-attached tape drive as an IBM zSystems tape drive, which provides a way to read/write “real” mainframe tape volumes. For specific drive and adapter details, see Chapter 14, “Tape drives and tapes” on page 303.
awsctc	Emulates an IBM 3088 channel-to-channel (CTC) adapter by using TCP/IP as the communication mechanism. The connection can be the same ISV zPDT instance, another instance in the same PC, or an ISV zPDT instance in a LAN-connected machine.

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

The current ISV zPDT design allows a maximum of 2048 emulated I/O devices, which are often described as 2048 *subchannels*. A practical number of emulated I/O devices is usually *much less* than 2048, depending on many factors.

2.2.3 Excluded IBM zSystems functions

Not all IBM zSystems instructions and functions are available with ISV zPDT. Instructions that are related to specific hardware facilities or optionally used by specialized programs might not be present. This excluded list includes these items:

- Base Control Program internal interface (BCPii) functions.
- List-directed initial program load (IPL) and Internal IPL.
- The accelerator; PKCS#11, EP11, and customized cryptographic routines (user-defined extensions (UDXs)) functions of cryptographic coprocessors; and IBM Trusted Key Entry (TKE) functions and interfaces.
- Time-of-day (TOD) steering.
- IBM zEnterprise® BladeCenter Extension (zBX) functions.
- CPU Measurement Facility or Hardware Instrumentation Services (HIS), including various counters and activities.
- Asynchronous data movers.
- IBM FICON®, and Transport Mode I/O.
- Parallel Access Volumes (PAVs).
- Logical channel subsystems.
- IBM HiperSockets functions.
- Logical partitions (LPARs).
- Functions involving the usage of Hardware System Area (HSA).
- Flash memory.
- Multiple I/O paths.
- Multithreading (MT) CPs (MT or symmetric multithreading (SMT)).
- Various fields within the Channel Measurement Facility reports.
- Hardware Management Consoles (HMCs).
- Some CHSC commands (such as crypto adapter measurements).
- System Recovery Boost.

- The DFLTCC asynchronous facility is missing, but the synchronous instruction is present. However, the feature indicator for the instruction is disabled. (It can be enabled with the zPDT command **`dflt`**.)
- The IBM S/390® Compatibility Mode function has limitations. For more information, see "S/390 Compatibility Mode" on page 274.
- IBM Secure Service Container (SSC) functions.
- OSA Address Table (OAT) configuration with the **`QUERYINFO`** command (for example, in z/OS).
- Dynamic sense for I/O operations in some cases, such as for mini-disks).
- The **`OSPROTECT=1`** function of z/OS. (**`OSPROTECT=SYSTEM`** is accepted.)
- Advanced I/O attachments or functions, such as IBM HyperSwap®.
- The Single System Image (SSI) and Live Guest Migration, as used by z/VM.
- ISV zPDT crypto master keys are not always usable between a previous zPDT release and a new zPDT release. In such cases, you must enter a new master key for the new zPDT release, and it can be the same key as used in the previous release.
- Various forms of sophisticated performance management, which are provided by complex instructions on recent IBM zSystems system processors, are not implemented by ISV zPDT. The actual instructions can be run (to avoid ABENDs), but they generally do nothing or effectively provide zero data.
- The zPDT emulation of 3590 tape drives has minor restrictions regarding the meaning of some sense data.
- Current releases of zPDT do not provide options to emulate older IBM architecture functions that changed in the architecture at the time of writing. For example, the OSAINFO function that was in OSA-Express3 is no longer provided.
- The IBM z16™ architecture offers certain "counters" reflecting the operation of selected features, such as some crypto operations. Such counters might not be present (or might not reflect appropriate values) during ISV zPDT operation.

IBM z15™ and later systems do not have zAAP specialty processors. zAAP specialty processors are mentioned throughout this publication for compatibility with earlier ISV zPDT releases.

2.2.4 ISV zPDT design and operational environments

The section lists several specific IBM zSystems features that are generally available with ISV zPDT. In addition to this list, ISV zPDT users should be aware of the general environments for which ISV zPDT is designed and tested, and what environments might not fit it well.

ISV zPDT is designed and tested to include the following environments:

- z/OS usage is accepted if the emulated memory for ISV zPDT is a reasonable size for the z/OS workload that is used. A 4 GB definition within the devmap for ISV zPDT might be for a minimal z/OS, and much larger sizes might be needed for reasonable performance of a particular workload.
- z/VM usage is accepted if the necessary memory sizes are present. For example, a z/VM plus two z/OS plus two CF images to produce an IBM Parallel Sysplex system is acceptable.
- z/VSE usage is accepted if necessary memory sizes are present. However, there is no formal zPDT support for z/VSE. For more information, contact your z/VSE provider.

- Linux for IBM zSystems (or Linux for S/390) is typically accepted if the necessary memory sizes are present. You (the ISV zPDT owner) must obtain, install, and configure the Linux for z package. IBM does not produce Linux for IBM zSystems. New releases sometimes contain unexpected installation “techniques” that might take time to resolve.
- Multiple ISV zPDT instances can be used, potentially with some shared emulated devices.
- Multiple concurrent users (for appropriate IBM zSystems software applications) are allowed and usually expected.
- A base Linux (to run ISV zPDT) that has sufficient memory (beyond the amount that is defined in ISV zPDT) to provide an effective Linux disk cache is important. This cache can be critical for reasonable performance by ISV zPDT.
- ISV zPDT supports “large memory” environments (1 MB and 2 GB pages for z/OS) and IBM zSystems system applications that use much memory. These environments can work well, but, especially with such usage, you must understand the performance implications of PC memory size, Linux disk cache usage, paging implications, and others.
- Operating in a virtual or container environment generally works if memory allocations are handled reasonably and the overall workload remains reasonable for the PC that is involved. LAN setup and operation can be more complex.
- The “internals” (hardware and firmware) for an IBM zSystems machine are complex to support complex environments with z/OS, z/VM, IBM z/Transaction Processing Facility (z/TPF), various Linux distributions for IBM zSystems, and others. ISV zPDT emulates much of this complexity when it is used in the environments that are listed here.
- ISV zPDT is generally not sensitive to the particular PC model that is used as the base, although a suitably configured high-end PC server can provide modest performance improvements and improved reliability.
- ISV zPDT is usually operated with a simple Linux desktop GUI for control and a relatively simple emulated 3270 window for a z/OS operator console. However, simple remote command-line interface (CLI) access to a PC server running ISV zPDT can be used instead of a GUI Linux console.
- The DFLTCC instruction is present as a synchronous instruction. The DFLTCC asynchronous function is not available in ISV zPDT. Most usages of the DFLTCC operation (by z/OS system code) detect and work with this configuration, but few exceptions exist. One known case involves early versions of z/OS 2.4 that result in 11E ABENDs followed by a wait state during a z/OS IPL. APAR fixes are available. One bypass is to use the zPDT **`dflt`** instruction after starting zPDT but before performing an IPL on z/OS.

ISV zPDT is not designed and tested to include the following areas; nevertheless, these options and directions might work for you. However, there is no unique ISV zPDT support for these areas, and you might need to address and resolve individual problems that you encounter.

- Many concurrent guest operating systems under z/VM can create issues. ISV zPDT design and testing usually stop at two z/OS systems (typically in a Parallel Sysplex environment) or two or three smaller operating systems. With larger environments, you must resolve any special problems that arise in your environment. Similar potential exposures exist with many virtual machines (VMs) (or containers) running ISV zPDT.
- IBM does not provide specific performance specifications for any version of ISV zPDT.
- Multiple ISV zPDT instances, when used excessively, can create system overloads that are difficult to resolve. At a basic level, the multiple instances have approximately the same considerations as multiple operating systems instances under z/VM or with containers.

- There is no formal testing or support for using ISV zPDT in a container. We are aware of many customers working in container environments and as best we understand there have been no basic problems. However, good planning is needed with special considerations for the amount of memory that is usable by zPDT and networking setups.
- There is no formal planning, testing, or support for using ISV zPDT in a cloud environment. We are aware of some customers doing so, but the ZPDT developers do not provide or recommend any details about such operations.
- Massive emulated disk operations can be a problem. Normal ISV zPDT operation typically involves a single emulated IBM zSystems channel. (OSA emulation uses extra emulated channels.) Excessive loads (as with too many separate operating system images under z/VM or through containers or other virtual environments) can result in significant I/O overloads and delays that can trigger watchdog timer issues or other complications. This situation is seldom a problem for “normal” ISV zPDT usage, but occurs in extreme situations.
- Java performance within ISV zPDT is slower than most other emulation functions, and extensive dependence on Java performance might not be the best option under current ISV zPDT versions.
- Some IBM zSystems I/O functions can create obscure problems. For example, converting z/OS volumes between z/VM mini-disks and “real” disks (both are zPDT emulated 3390 disks) involves thorough understanding of many rules and conventions to avoid obscure issues.[5] Also, projects that require alterations to the emulated cryptographic adapter are not generally supported. In particular, TKE operation is not supported.
- Multiple concurrent users (such as Time Sharing Option (TSO) users) are permitted, but the general design is for “some” with whatever limitations are appropriate for the hardware and the software configuration. Dozens of concurrent users are not considered to be within the general ISV zPDT sphere, although the specific numbers that are involved depend on many factors.
- The ISV zPDT license typically includes copying a particular z/OS system (the Application Development Controlled Distribution (ADCD) z/OS system) that is prepared by IBM. (A z/VM system is also available for ISV zPDT users.) Any other IBM software is not included with ISV zPDT, and exceptions must be discussed with an IBM representative. (Each IBM ZD&T license contract might be different.)
- Complex virtual or container environments can have many issues, especially for memory allocation, networking, or shared I/O accesses. Overloading the base system can create various problems. Although there are many zPDT owners operating in these environments, the setup skills are up to the zPDT owner, and they can be more complex than anticipated.

  ISV zPDT is tested with some of the more basic virtual and container environments, but not with all versions, and there is a growing number of “other” versions and options. This area can be more complex than expected. In rare cases, we have seen zPDT errors such as “Processor family not supported” when running in “other” virtual or container environments. More often, we hear about performance problems that often are related to the amount of memory or the usage if it in the zPDT environment.

[5] One problem element, which is related to I/O dynamic sensing, was recently bypassed with ISV zPDT.

- When started, ISV zPDT coordinates TOD with the underlying Linux (with some leap second adjustment), but then manages its various internal clocks and timers separately.[6] Extensive and substantial system loads or “production-like” intervals between ISV zPDT restarts can skew these clocks or times to the point where the ISV zPDT license is invalid and an ISV zPDT restart is needed.
- The ISV zPDT package (with the normal ADCD z/OS system) is not designed for people unfamiliar with IBM zSystems machines and z/OS. Most ISV zPDT users are familiar with these topics. The basic ISV zPDT *can* be used in a learning environment, although a simpler, more basic z/OS package is appropriate, as is appropriate “educational” documentation, monitoring, and support.
- ISV zPDT development and testing are done with several different PC Linux distributions. There is no testing (or design) suitable for *all* Linux releases and versions. ISV zPDT is assumed to be the major function on its base Linux PC, and it is better to avoid using “other” Linux versions or in some cases the most frequent updates. A stable base is a better goal. The Linux distributions and levels that are used for ISV zPDT releases are briefly listed in 2.5, “ISV zPDT releases” on page 44.

  New Linux distributions or updates sometimes change the default installed library modules and other factors, and the ISV zPDT developers cannot always foresee these changes. One solution is to stay with the Linux distributions and levels that are documented for your ISV zPDT release. Another solution is to observe and search for various internet discussions about new Linux versions.

  New Linux distribution levels sometimes change setup options, the configuration controls, or the firewall functions that are used for network usage. An existing ISV zPDT release might require user assistance when encountering these issues.
- Terminal connections (through emulated OSA or emulated IBM 3274 interfaces) depend on ISV zPDT operation, base Linux operation and configuration, and the IBM zSystems operating system parameters for the links that are involved. This situation can become complex, and solutions depend on user or installation skills.
- IBM has not tested extreme ISV zPDT configurations. For example, in theory an ISV zPDT instance can have up to 2048 devices, up to 8 CPs, and base Linux can have up to 15 concurrent ISV zPDT instances. In a wildly extreme configuration, this setup might represent 15 * 2048 = 30,000 emulated devices being used by 120 CPs. Extreme configurations, even though much smaller than this example, might not be practical. Among other considerations, each emulated device requires control blocks in Linux shared memory, and a large configuration might cause difficulties with Linux shared memory and swap file configurations.
- ISV zPDT performance is partly determined by the effectiveness of the Linux disk cache, and ISV zPDT cannot directly manage this aspect of normal Linux operation. The effectiveness of the Linux disk cache can change as workload characteristics change, and it is determined by many factors. It is easier to understand in simple cases where it is not affected by virtual operations, multiple IBM zSystems operating systems, or container usage.
- At the time of writing, network-attached storage (NAS) disks had mixed reviews by ISV zPDT users. The issues are at the Linux level. ISV zPDT is unaware of the nature of the Linux disks except when access delays are so extensive that z/OS timeouts are triggered. If Linux detects I/O problems with fairly intensive usage of the NAS disks, they are not appropriate for ISV zPDT use. In some cases, the usability might be related to the congestion and bandwidth of the LAN that is involved.

[6] Date and time values in Linux can be handled in many ways, which is seldom a problem in a single ISV zPDT system, but if several ISV zPDT systems are linked together (on separate PCs), time management on the different Linux PCs might need to be done in a unified manner.

- Linux root authority is needed to install ISV zPDT and run a few of the administrative commands. Once ISV zPDT is installed, the release of ISV zPDT at the time of writing (GA10.3 and later) does not need base Linux root access for normal operation.
- New releases of Linux for S/390 (or for IBM zSystems) sometimes have differing technology that can cause problems with ISV zPDT. For example, at the time of writing, a particular new version fails when installing it directly on ISV zPDT, but it can be installed under z/VM. This particular problem probably will be resolved by the time you read this publication, but other similar problems might be in future releases. Future ISV zPDT releases can address some of these problems, but there is usually a time gap between the occurrence of such problems and a future ISV zPDT release or fix pack that addresses some such problems.
- zPDT does not specify how you might arrange emulated volumes (and other files that zPDT might access) within your Linux environment, but zPDT must have at least *read* access to whatever Linux directories are involved. In a realistic sense, zPDT probably should have *read/write* access to such directories and the files representing emulated volumes.[7]
- z/OS is not an integrated part of ISV zPDT, but the ADCD implementation of zPDT is often included in an ISV zPDT product package. z/OS questions and issues are not a direct part of ISV zPDT support, and some issues might require more user administration skills. For example, the ADCD z/OS package uses RACF for security controls. Usage of other security controls instead of RACF is an example that can introduce additional administration effects.

2.2.5 ISV zPDT security, integrity, and RAS concepts

ISV zPDT emulates IBM zSystems architecture while running as a PC Linux application. ISV zPDT has no control over the security or integrity environment of this “base” Linux. Although ISV zPDT generally follows reasonable Linux application standards, ISV zPDT should *not* be considered a secure system unless all aspects of access to the base Linux are also considered. Within ISV zPDT itself (while running an IBM operating system), the normal security and integrity of that environment exists.[8] For example, the z/OS ADCD package that is typically available to ISV zPDT users contains IBM RACF, which can be used to manage security within the z/OS environment.

At the base Linux level, there is potential for many exposures. For example, a root user can inspect any emulated 3390 or emulated tape file and potentially uncover confidential data that is stored in these emulated devices. A malicious Linux user might “front end” various ISV zPDT administrative commands (which run as ordinary Linux commands), although ISV zPDT provides some protection against this action.

Ideally, access to the base Linux system running ISV zPDT might be limited to only the necessary trusted administrative personnel. General user access to the IBM zSystems operating system running under ISV zPDT would be only through IBM zSystems interfaces, such as emulated 3270 terminals.[9] Such a restrictive environment is not always possible, and even where this environment is intended, skills are needed to create and maintain it.

[7] Intentional planning for read-only emulated volumes is an exception for requiring *write* access.
[8] A notable exception is that cryptographic keys for emulated cryptographic adapters are stored in standard Linux files. These files are secure from the IBM zSystems viewpoint, but not from the Linux viewpoint.
[9] The awscmd device manager provides a method for z/OS or IBM z/VM users to send commands to the base Linux. This function is useful to some ISV zPDT customers, but might provide a security exposure for other customers.

ISV zPDT does *not* provide the RAS of a standard IBM zSystems system. ISV zPDT has no control over exposures in the underlying Linux system or the underlying PC system. Careful selection, configuration, and management of these elements can produce a good system, but there is no claim that it equals the RAS of a standard IBM zSystems system. Prudent users should have defined backup procedures (for emulated volumes), procedures for monitoring Linux and z/OS consoles (for error messages), periodic checking of emulated volume structure validity (by using the **`alcckd -rs`** command, among other tools), and monitoring to verify that the system is not routinely overloaded.

zPDT can emulate some features of the IBM zSystems system cryptographic adapter, including the use of "master keys." These (emulated) master keys are stored in a normal base Linux file, which is *not* secure.

Important: ISV zPDT is intended for development work. It is not intended as a secure system. Think carefully before moving any confidential data to your ISV zPDT systems. You should not enter a "real" master key (as used on a real IBM zSystems system) in a zPDT system without fully understanding the potential exposures that are involved.

2.2.6 Performance

IBM does not provide performance or capacity specifications for ISV zPDT. Specifying performance or capacity for ISV zPDT is too difficult for many reasons, including the following ones:

- Performance depends on the power of the underlying hardware, which changes frequently. Performance is related to the clock speed of the underlying processor (such as 3.4 GHz for an Intel processor) and the memory design, pipelining, caching, instruction availability, and translation design of the underlying processor.
- Linux performance (including applications such as ISV zPDT) can be greatly influenced by how the Linux disk cache (and swap file) is performing, and the nature of the Linux disks.
- The number of IBM zSystems CPs that is emulated by ISV zPDT has an obvious effect, as do the number of cores in the PC processor, but the effect is not linear.
- Every new release or update of ISV zPDT can change performance.
- The IBM zSystems instruction mix and memory reference pattern has a profound impact on performance, which is a greater impact than is observed on a larger IBM zSystems system.
- Million instructions per second (MIPS) is a rather discredited metric, although it is still informally used with smaller IBM zSystems machines. Any MIPS number for ISV zPDT is *very* dependent on the nature of the workload and the Linux configuration.
- I/O performance must be considered. For example, all emulated disk and tape operations for ISV zPDT might be from a single (relatively slow) computer disk drive or solid-state drives (SSDs). Workloads with modest I/O loads (when run on a larger IBM zSystems system) might be I/O-bound on an ISV zPDT system.
- Virtualization or containers can add much more variability, especially when the host computer is overcommitted.

z/VM can be used with ISV zPDT. The performance of guest operating systems under z/VM (such as z/OS running under z/VM) is influenced by the usage of the START INTERPRETIVE EXECUTION (SIE) function. On a large IBM zSystems machine, this function provides a "microcode assist"[10] for many of the virtualization functions that are performed by z/VM. Most SIE functions are provided by ISV zPDT, but there is no direct equivalent of a "microcode assist" level, and the virtualization performance boost that is provided by SIE is modest.

In most cases, ISV zPDT performance partly depends on the JIT operation that is employed by the ISV zPDT emulator.[11] This function is internal to ISV zPDT and is not documented other than this brief note. The function is, in a sense, a parallel operation to the simple emulation of each IBM zSystems instruction. This parallel function can detect reasonable IBM zSystems instruction loops. When the loops are detected, the instructions in the loop are forwarded to the JIT "dynamic compiler", where they are consolidated into an optimized string of Intel compatible instructions. This optimized string is run instead of using individual emulation of each IBM zSystems instruction that is encountered in the loop. The implementation is complex, and it tends to be updated with each new ISV zPDT release.

ISV zPDT is not intended to replace normal IBM zSystems configurations. If you are considering a configuration with more than, for example, a hundred emulated devices, with heavily loaded I/O devices, or with many z/VM guests, discuss your requirements with your ISV zPDT supplier. Moderately large configurations *are* possible and can be acceptable, but you should review your plans with knowledgeable ISV zPDT people. The key to understanding ISV zPDT capacity and performance is a *detailed* understanding of your workload.

2.2.7 IBM zSystems architecture levels

IBM zSystems machines have architectural characteristics, such as new instructions on newer systems or changed firmware characteristics. The architectures are not always compatible with earlier versions, and might require software updates to run older operating systems on newer architectures. For example, z15 machines (including ISV zPDT GA10) do not run with the original z/VM 6.2 release. In this case, a program temporary fix (PTF) is needed to resolve the incompatibility. Such software updates are often known as "toleration PTFs", and they are usually available for some older operating system releases when a new IBM zSystems series becomes available. IBM does not provide such updates for much earlier operating systems.

The IBM zSystems architecture levels for the ISV zPDT CPs are shown in Table 2-1.

Table 2-1 IBM zSystems architecture levels

Release date	ISV zPDT release	ISV zPDT build level	IBM zSystems architecture	ARCH level (for compilers)
2009 and 2010	V1R1 "GA1"	39.xx	z800 and z900	ARCH(7)
2011	V1R2 "GA2"	41.xx	IBM z10	ARCH(8)
2012	V1R3 "GA3"	43.xx	z196	ARCH(9)
2013	V1R4 "GA4"	45.xx	EC 12	ARCH(10)
2014	V1R5 "GA5"	47.xx	EC 12 GA 2	ARCH(10)

[10] This term is the common one for SIE operations, although the actual implementation might be much more complex than implied by this statement.

[11] JIT is often connected to Java operation, which is not the case here, where JIT is used as a more generalized term. Also, JIT is undergoing changes at the time of writing.

Release date	ISV zPDT release	ISV zPDT build level	IBM zSystems architecture	ARCH level (for compilers)
2015	V1R6 “GA6”	49.xx	IBM z13®	ARCH(11)
1Q 2017	V1R7 “GA7”	49.xx	z13 GA2	ARCH(11)
4Q 2017	V1R8 “GA8”	51.xx	IBM z14	ARCH(12)
2Q2019	V1R9 “GA 9”	53.xx	IBM z14 GA2	ARCH(12)
1Q2020	V1R10 “GA10”	55.xx	IBM z15	ARCH(13)
3Q2022	V1R11 “GA11”	57.xx	z16	ARCH(14)

ISV zPDT does not have a facility to emulate older IBM zSystems architectures. For example, the current release (ISV zPDT 1.11) is at the z16 level. It cannot be set to an IBM zSystems 196 level or a z10 level, for example. Providing a switchable architectural level facility results in reduced performance.

If you want to test software on older IBM zSystems architectures (and older z/OS releases), you must retain older versions of ISV zPDT. Older ISV zPDT releases might or might not work correctly with the latest Linux distributions, and IBM cannot help in this area. In general, you must retain older PC hardware, older Linux releases, older z/OS releases, and older ISV zPDT releases if you want to consistently run your software in older operating environments. IBM does not have a way to distribute older ISV zPDT releases or older ADCD releases.

2.2.8 Virtualization and containers

ISV zPDT can be used in a virtual environment or in Docker containers. As a best practice, you should have some experience with ISV zPDT (and whatever operating systems are used under it) in a more basic environment before attempting to use ISV zPDT in a virtual environment or Docker containers.

The most common performance problem that we encountered with virtual or container environments is constrained memory and memory planning.

Docker

IBM has done basic testing of ISV zPDT running in a Docker environment, which can be a workable environment for ISV zPDT operation. As with other types of virtualization, we caution against over-commitment of the base system. These cautions are related to memory sizes, availability, and system overload.

Simple Docker usage involves applications that are represented by a single Linux process. ISV zPDT operation has many Linux processes; furthermore, ISV zPDT operation requires the availability of auxiliary Linux programs such as the awsckd and aws3274 device managers, and the `awsstart` and `ipl` commands. The network configuration can be especially complex to implement.

ISV zPDT does not provide a sample Docker script or image. If you build your own image, you should consider the following goals:

- The Linux CLI that is used to start ISV zPDT (the 3270 window for the z/OS master console and a 3270 window for TSO usage) should be available in some manner outside the immediate Docker environment. Likewise, the ISV zPDT logs directory and files must be available for debugging.
- Any ISV zPDT core image files should be exposed to the base Linux. They might need to be examined or retained for debugging assistance.
- The need for Linux root authority is limited in basic ISV zPDT operation. However, your Docker configuration and parameters might increase the need for Linux root authority when running ISV zPDT.

Virtual environments

The VMware, Kernel-based Virtual Machine (KVM), and Xen virtual environments are sometimes used during ISV zPDT development, although they are not the basic focus of ISV zPDT. Other virtual environments might operate correctly, but have not been tested by ISV zPDT developers.

ISV zPDT (running z/OS) is typically a “heavy” workload. We strongly advise that you do not run ISV zPDT in overcommitted virtual environments. Among other effects, a substantially overcommitted virtual server might cause delays that trigger z/OS missing interrupt handler or SPINLOOP warnings. Another danger might be cascading page faults, where the VM hypervisor and the guest Linux running ISV zPDT and z/OS might all be paging due to several levels of overcommitted memory.

You (the ISV zPDT owner, user, or administrator) must obtain the necessary skills to install, configure, and use your virtual server. We do not attempt to document or provide instructions about installing and managing the virtual server environment. We encountered several cases of virtual ISV zPDT operation in servers that are managed by staff who are not familiar with the workload characteristics of ISV zPDT, z/OS, and others. In some cases, the server is overcommitted, with the server staff commenting, “We always plan this way”. Insufficient resources for ISV zPDT can produce many problems that are difficult to diagnose.

The tested virtual environments normally use remote ISV zPDT license and Unique Identity Manager (UIM) servers. These servers might not be necessary for smaller configurations where a separate USB port (for a zPDT license token) could be assigned to each virtual guest ISV zPDT. A single ISV zPDT token (connected to a USB port) typically cannot be shared by multiple VMs. In general, the first VM that starts (that specifies USB usage) occupies the USB interface to the token. Using a remote license server allows a token to be shared by multiple virtual ISV zPDT instances (assuming the token can provide sufficient licenses).

It is possible to configure logical disk drives to be shared among multiple virtual guest machines. *Do not do take this action unless you are certain that you know what you are doing*. Linux (which we assume is the basic operating system on all the VMs) does not routinely support shared disks.

We found that z/OS guests in a virtual environment have performance ranging from excellent to unacceptable, depending on the nature of the workload and whether the server was overcommitted in some way. A virtualized environment cannot create more machine capacity than what exists in the underlying hardware. The key consideration is the nature of the workloads.

The focus in this publication is z/OS, but this focus does not imply any particular workload under z/OS. A z/OS system with many TSO users (mostly editing source code or doing occasional compilations) might be considered lightly loaded, and another z/OS system with only a few users running large Db2 or Java jobs might be heavily loaded. You cannot draw any conclusions about performance unless you can realistically understand your workloads.

2.2.9 Hardware tokens

The hardware tokens at the time of writing are shown in Figure 2-1.

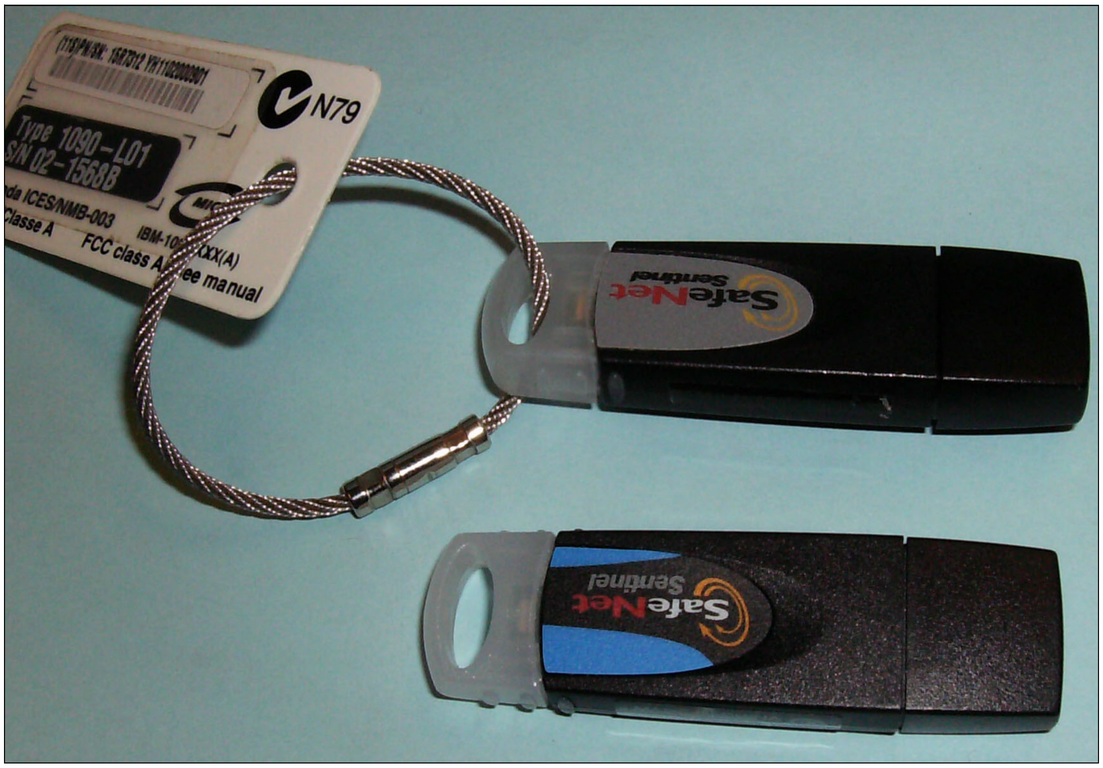


Figure 2-1 The 1090 (top one in photo) and 1091 hardware keys

These tokens are the IBM 1090 (for ISV zPDT) and IBM 1091 (for IBM ZD&T) tokens. Here are the specific “token” aspects:

- An ISV zPDT hardware token is a USB device that resembles a typical USB flash drive, and it must be installed in a USB port for ISV zPDT to be operational.
- The hardware tokens that are shown are Generation 1 tokens. At the time of writing, these tokens are the only hardware tokens that are available. Generation 2 hardware tokens are briefly described in Appendix C, “Generation 2 tokens and licenses” on page 379, and they might be available in the future.
- An ISV zPDT hardware token can be installed in a USB port of the PC that is running ISV zPDT or it can be installed in a “remote license server” that has a network connection to the PC that is running ISV zPDT. The remote hardware token server also has the ISV zPDT product that is installed to provide the token interface and the network interface for ISV zPDT operation. A remote license server can provide more security for the hardware token itself and ISV zPDT licenses for multiple operational ISV zPDT systems if sufficient licenses are available on the token.

- Generation 1 hardware tokens require Linux “drivers” (furnished by the token vendor) that are 32-bit Linux functions. It might be necessary to take special steps with a Linux installation to find the 32-bit Linux libraries that are needed. These “drivers” are packaged with ISV zPDT and *only* these versions can be used.
- The current ISV zPDT tokens contain one, two, or three licenses, and they have numbers such as 1090-L01, 1090-L02, and 1090-L03. The number of licenses corresponds to the number of IBM zSystems CPs that can be concurrently emulated. The licenses are typically good for 1 year, and they can be renewed through your ISV zPDT vendor.
- Multiple ISV zPDT hardware tokens can be installed on the PC to obtain more licenses. For example, using two 1090-L03 tokens provides up to six CPs. The maximum number of CPs that is supported for a single instance of ISV zPDT operation is eight (including zIIP, zAAP, and IFL processors, but not counting CFs).
- Starting with ISV zPDT GA8, zIIPs do not “count” when considering the number of licenses in your tokens, although they do count toward the maximum number of CPs. For more information, see 13.1, “Minor ISV zPDT notes” on page 274.
- When “token” is mentioned in this publication, we are referring to 1090 tokens (for ISV zPDT). The token can be installed on the PC running ISV zPDT, or this PC can be connected to a remote ISV zPDT license server.
- A 1090 token should always have an attached tag, as shown in Figure 2-1 on page 33. This tag contains serial numbers that are needed to renew the ISV zPDT licenses in the token. (IBM ZD&T tokens have a serial number that is engraved on the back of the token.)
- A Generation 2 “software” license is available for IBM ZD&T usage, which is briefly described in Appendix C, “Generation 2 tokens and licenses” on page 379 and Appendix D, “IBM Z Development and Test Environment notes” on page 387. This license is more generally described in IBM ZD&T documentation.

If the token is removed while ISV zPDT is operational or if the connection to a remote license server is lost, the operation pauses with a series of messages. If the intervening time interval does not disrupt the operating system or application programs, ISV zPDT operation can be resumed by connecting the license again.

An ISV zPDT USB token is normally valid for 1 year after it is initialized or *activated*. It can be reinitialized[12] at any time, which normally extends the validity for 1 year beyond the date of the most recent reinitialization.[13] The procedure for initializing the key (or reinitializing it) depends on the channel that you used to obtain your ISV zPDT system, such as an IBM Business Partner or other supplier. For more information about token updating, see 8.11, “Token activation and renewal” on page 196.

Various “SMP effects” reduce the effectiveness of extra CPs. For example, going from seven to eight CPs might offer minimal performance enhancements for many workloads. The I/O capability of the underlying PC must also be considered. However, this performance determination is left to you.

[12] This action is also known as a “lease extension.”
[13] The extension period might differ depending on the IBM channel that was used to obtain the ISV zPDT system.

2.3 PC selection overview for ISV zPDT

Various PCs can be chosen for ISV zPDT operation if you consider some basic parameters.

ISV zPDT is generally not sensitive to the brand or model of the underlying PC. We see systems ranging from rather old laptops (two cores with 8 GB of memory) to large servers (24 cores or more with up to 256 GB or more memory). Considerations for selecting a PC include the following items:

- Is absolute peak performance required? The fastest machine that we have seen (at the time of writing) in terms of raw emulated CPU performance is a large server. In second place is a high-end laptop. However, the performance differences among recent high-end PCs (assuming they are reasonably configured for the workload involved) are not great.
- How important is reliability? Elements include RAID, dual power supplies, memory recovery technology, inclusion of support processor functions, battery backup, and others.
- Do you need a graphic desktop? Laptops are good at GUIs, but servers are often rack-mounted in an "operations" area without a good display, keyboard, or mouse. A remote CLI connection to your zPDT system (probably by using SSH) can be used for zPDT administration.
- How much memory do you need? This topic can be complex. In this publication, we recommend at least 1 GB more than the emulated IBM zSystems size, but at a minimum. The Linux disk cache is an important performance element that depends on sufficient memory. Likewise, the ability to avoid Linux swapping is important for performance. As a starting point, use *at least* 8 GB more than the zPDT memory that is defined in the devmap. Typical ISV zPDT systems might have a PC memory size at least twice as large as the defined ISV zPDT IBM zSystems memory.
- How many PC cores do you need? ISV zPDT requires a number of cores that is at least equal to the number of ISV zPDT CPs, including zIIPs, zAAPs, and IFLs. Again, this number is the minimum, and two or three more cores can improve performance. If significant additional workloads are present in Linux, then the number of cores that is needed is likely much larger.
- Do you plan a virtual or container environment? These items always have some performance implications, but can have a major impact if the server is over-committed or if memory usage is not planned.
- Does your z/OS system (including expected applications and subsystems) need a large amount of memory? A z/OS system with a number of TSO users editing and submitting compilations might perform better (that is, not paging) with 8 GB of memory, but a large IBM Db2 workload that uses several buffers might perform better with 100+ GB of memory. We observed a tendency to overestimate the z/OS memory that is needed (in the devmap), and underestimate the importance of memory for Linux. Available PC memory is not dedicated to either ISV zPDT or the disk cache, but is managed by Linux.
- Will your terminal users be on a local network or the internet? Local networks (not directly connected to the internet) typically have IP addresses of `192.168.x.x` or `10.x.x.x` and allow local management of specific IP addresses. Local networks often allow general routing that effectively interconnects all the PCs on the network. Conversely, connections to the internet typically require a specially assigned IP address that is known to be on specific routing throughout the internet. Such IP addresses tend to be not portable. (DHCP addresses for a PC running ISV zPDT are alternative options for internet connections, but tend to be impractical for external users wanting to connect to the ISV zPDT system.)

Base PC hardware notes

IBM uses various PCs for core ISV zPDT development and testing. It is not possible to list all the hardware that is used, but practical notes on the hardware include the following items:

- In all test cases, a *minimum* of 16 GB PC of memory was available. Systems with up to 256 GB were used.
- Hardware RAID adapters are used for most systems above the laptop level, especially when the systems are rack-mounted in areas where there is not much visual inspection. Furthermore, routine verification of the "normal" RAID operation is suggested.
- Even for rack-mounted servers, a GUI display and keyboard (often mounted on a movable cart) are available when needed.
- In cases where many PCs are involved, the ISV zPDT tokens are installed in remote ISV zPDT license servers. (*Remote* often means in the same floor area and not distantly remote.) These cases typically require an additional PC for the license server and a reliable local network.
- A suitable USB port must be available for the hardware tokens. This item applies whether tokens are used on the base ISV zPDT machine or on a remote license server. Do *not* use an unpowered USB port expander for ISV zPDT tokens.
- Multiple LAN interfaces might be useful in larger configurations, although this situation is rare. Do not use multiple LAN interfaces unless you know that they are needed for a specific reason.
- Disable hyperthreading (if available) at the BIOS level. Hyperthreading can produce slowdowns when z/OS is running due to spinloops. If many PC cores are available, the slowdowns might be resolved before z/OS console messages are produced, which means that there is no indication of a problem other than reduced performance. For more information, see 13.1, "Minor ISV zPDT notes" on page 274.
- The Linux distribution must operate correctly on the base PC. New adapters, various power management options, new USB chips, new display parameters, new disk technology, and other technology-related items might not work correctly with all Linux distributions or might require extra Linux device drivers or Linux updates.
- Some SCSI tape drives can be used with ISV zPDT, but not all SCSI tape drives are usable by ISV zPDT. The usability depends on the exact model, the exact firmware level, the exact SCSI adapter that is used, and the firmware options that are set in the drive. IBM cannot predict whether *your* SCSI drive works with ISV zPDT. If this item is important to you, discuss your requirements with your ISV zPDT provider. For more information, see Chapter 14, "Tape drives and tapes" on page 303.
- Although PC hardware incompatibility has not been extensively explored by IBM, it is rare. The only two cases that are known at the time of writing are related to old PCs (preventing any ISV zPDT operation), and a problem with crypto adapter emulation on an older machine (since resolved with an ISV zPDT update).
- Although some ISV zPDT testing is done with virtual or container environments, this testing does not attempt to explore all the possible alternatives in these areas. You must have the skills to explore and solve unexpected problems in these environments.

2.3.1 zPDT and PC memory

The complete ISV zPDT memory environment exists in Linux virtual memory. Linux is aggressive in allocating real PC memory frames to virtual memory pages and disk file data by using its own judgment about what is the best usage of real memory. The situation is variable when Linux caching of disk I/O is considered, and disk caching is an important element of Linux performance.

ISV zPDT exists in Linux virtual memory. We might informally say something like, “With an 8 GB machine, we can allow1 GB for Linux and 7 GB for ISV zPDT,” but such statements must *not* be taken *literally*. ISV zPDT does not physically partition PC memory. If we inspect the machine in this example at a random time, we might find 1.2 GB of memory that is owned by the primary ISV zPDT module, 0.2 GB of memory that is owned by recognizable core Linux functions, 3.8 GB of memory that is used for disk data cache, 0.2 of memory that is used by various other processes (such as ISV zPDT device managers), and the rest unassigned. A few seconds later, the usage statistics might be different.

The PC memory size should be substantially larger than the sum of all concurrent ISV zPDT defined IBM zSystems memory. More is usually better. An arrangement might have PC memory at least twice the size of the defined IBM zSystems memory. The primary goals are to avoid Linux paging that stalls ISV zPDT operation and 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. This management is often overlooked for virtual or container environments, which results in unexpected poor performance. Some practical experimentation, based on your configurations and your workloads, might be needed.

2.3.2 PC disk space

The disk space for the ISV zPDT executable programs and control files is relatively small.[14] The disk space for emulated IBM zSystems volumes is not small, so some planning is needed. The space for emulated disk volumes can be calculated accurately, but the space for emulated tape volumes depends 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 required space is shown in Table 2-2.

Table 2-2 Required disk space for 3390 emulated disk volumes

3390 model	Approximate space that is required	Exact space that is required
3390-1	0.95 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

One 3390 cylinder is equal to 852,480[15] bytes.

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

Disk space is needed for 3390 volumes that contain the operating system, and for whatever local data volumes that you might produce. For example, the z/OS ADCD system, at the time of writing, needs 112 GB - 224 GB, depending on which volumes you decide to include.

Emulated tape sizes reflect the size of the data that is written on the tape with a small, additional space (less than 1%) that is needed for awstape control blocks.[16] (Optionally, the awstape device manager can compress these files, which can reduce the amount of space that is used.)

[14] It is typically less than about 70 MB.
[15] Each volume has an extra 512 bytes overhead.
[16] The actual overhead is 6 bytes for each block that is written (including a tape mark, which counts as a block).

2.3.3 PC model information for IBM

The ISV zPDT formal IBM license statement regarding base systems includes the following text:

"The Program may be used on the following systems that are running versions of Linux as specified in the Program's read-me file: IBM System x 3500 M1, 3500 M2, 3500 M3, 3500 M4, 3650 M1, 3650 M2, 3650 M3, or 3650 M4; Lenovo Thinkpad W Series; or systems that are otherwise approved by IBM."

The license agreements might contain reporting requirements that must be understood by the user. These requirements are not covered in this publication, but they can be reviewed with your IBM representative or ISV zPDT supplier. (As a practical matter, most ISV zPDT operation is on later systems than the ones that are listed here.)

2.4 Practical ISV zPDT operational notes

A number of general concepts that are involved in ISV zPDT usage are described in this section. Specific instructions about ISV zPDT installation, operation, and management are provided in later chapters in this publication.

2.4.1 PC software levels

Both PC hardware and base Linux software change frequently. ISV zPDT changes are needed to maintain a reasonable level of compatibility. ISV zPDT is not intended to be compatible with all levels of Linux or with all available PC hardware.

Base Linux

At the time of writing, ISV zPDT is built for operation on the Linux levels that are listed in 2.5.1, "Current release" on page 44. These levels are the "supported" base Linux releases. Earlier Linux distributions should not be used due to potential Linux library differences. Various Linux distributions and levels might require you to make administrative adjustments. For example, at the time of writing, some Linux distributions require additional commands to provide reasonable OSA performance.

Important: Over time, ISV zPDT will follow general Linux developments and changes, but constantly following the latest Linux distributions and updates is not a primary ISV zPDT goal.

Do *not* confuse the following two Linux distributions:

- The Linux that you install on your PC to run ISV zPDT, which is your *base Linux*.
- The Linux for IBM zSystems (or Linux for S/390) distribution that you might run under ISV zPDT.

These two distributions are separate topics. With few exceptions, all mentions of "Linux" in this publication refer to the Linux that you install on your PC.

3270 emulator

A suitable 3270 emulator is usually needed, but it is not distributed with ISV zPDT. The most common 3270 emulator for Linux is x3270. Some current Linux distributions might not include the x3270 package, but it can be downloaded from various sites. Other 3270 emulators might be used, but their operation with ISV zPDT must be checked by you. IBM developers have used recent releases of the IBM Personal Communications package (on Microsoft Windows systems).

2.4.2 Linux user IDs

In principle, any Linux user ID can be used to install[17] or operate ISV zPDT, with the exception that an ISV zPDT operational Linux user ID cannot be longer than 8 characters (unless the **`system_name`** statement is used in the devmap, in which case there is no special limit on the Linux user ID length). All our examples assume user ID `ibmsys1` is used, but there is nothing special about this name. The ISV zPDT system uses several default path names that are related to the current Linux user ID.

In principle, a different Linux user ID can be used to create a different ISV zPDT operational environment with different control files. Also, multiple Linux user IDs *must* be used when running multiple ISV zPDT instances concurrently. We use `ibmsys2` and `ibmsys3` as examples of these additional user IDs.

Current Linux operating systems automatically create home directories for user IDs in `/home/<user ID>`. For example, the home directory for user ID `ibmsys1` is `/home/ibmsys1`. It is possible to specify a different home directory for a user ID. Throughout this publication, we often use `/home/ibmsys1` to indicate the home directory for the ISV zPDT user ID, even though the specific usage of `ibmsys1` is not required.

2.4.3 ISV zPDT operational components

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

- A base Linux system, which is not provided with ISV zPDT. You must acquire it directly.
- A suitable 3270 emulator (which is usually run on the same PC that is hosting ISV zPDT, although this setup is not required). The ISV zPDT package does not provide a 3270 emulator.
- A hardware USB token, which is required for ISV zPDT operation.
- An ISV zPDT program package file. Within this single file are the following items:
  - Two prerequisite SafeNet driver programs for communicating with the token. These two drivers are provided with ISV zPDT and only these provided versions can be used. Other versions that are available from the web must *not* be used even if they appear to be a later level. These two programs are installed even if a remote license server is used.
  - A program for communicating with the license servers.
  - The Red Hat Enterprise Linux (RHEL) version of ISV zPDT.
  - The Novell (SUSE Linux Enterprise Server) version of ISV zPDT.
  - The Ubuntu version of ISV zPDT.

[17] Part of the installation process must be done as root, but the initial login should be with the user ID that will be used to operate ISV zPDT.

– An installer program that displays a license, installs the prerequisite drivers (if not already present), and then selects and installs the correct ISV zPDT version.

– Components that provide remote license and identity management functions.

IBM zSystems software, such as z/OS, is not part of ISV zPDT, although it is typically included in the IBM product package that includes ISV zPDT.

With only a few exceptions (usually dealing with installation), ISV zPDT discussions are the same whether the Red Hat, Novell, or Ubuntu distributions are used. Different versions of ISV zPDT (for Red Hat, Novell, and Ubuntu) are provided within the ISV zPDT package due to slightly different library levels or contents in these three environments.

The ISV zPDT installation also creates the `/usr/z1090/man` and `/usr/z1090/uim` directories. The `uim` directory contains several small files that are used to provide a consistent serial number for IBM zSystems compatibility. The `man` directory contains normal Linux `man` pages for ISV zPDT.

The first start of ISV zPDT creates several subdirectories (that are placed in the z1090 subdirectory) in the user's home directory.[18] Briefly, these subdirectories are as follows:

- `cards`, `lists`: Can be used to provide input files to an emulated card reader or output from an emulated printer.[19] If not used, they are empty. Few ISV zPDT customers use these files.
- `disks`, `tapes`: Can 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 ISV zPDT to hold various dumps, logs, and traces. ISV zPDT partly manages the contents of this subdirectory. The contents of this directory are important if it becomes necessary to investigate an ISV zPDT failure.
- `configs`, `pipes`, `srdis`: Used for ISV zPDT internal processing. Do not erase or alter the contents of these small subdirectories.

Minor usage of the `/tmp` file system occurs during ISV zPDT installation, ADCD installation, aws3270 device manager start, and optionally for Server Time Protocol (STP) logs.

2.4.4 ISV zPDT console

An ISV zPDT system is partly administered from Linux CLIs. This operation can be done remotely through telnet or SSH connections. A graphics connection is not needed.

Do *not* confuse the MVS operator console, for example, running in a 3270 window with Linux terminal commands that are used to administer ISV zPDT, which are run from a Linux CLI. We are describing zPDT administrative commands, not z/OS operator commands.

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

[18] When ISV zPDT starts, a z1090 subdirectory is created in the home directory of the user (if it does not exist). The subdirectories that are described here are under the z1090 subdirectory.

[19] These directories are not required for emulated card reader or printer operation.

ISV zPDT sometimes issues asynchronous messages, which are sent to the Linux CLI that was used to start that ISV zPDT instance. If that CLI is closed, the asynchronous messages are not seen.[20] You can issue ISV zPDT commands from any Linux CLI running under the Linux user ID that started the ISV zPDT instance.

2.4.5 ISV zPDT device maps

A *devmap* is a simple Linux text file. You might have many devmaps, each of which are a separate Linux file. A devmap is specified when ISV zPDT starts. You may use a different devmap each time an ISV zPDT instance starts. A devmap specifies the IBM zSystems characteristics to use and the device managers (with their parameters) to use for an instance of ISV zPDT operation.

The following devmap illustrates a simple IBM zSystems configuration:

```
[system]
memory 8000m                    # emulated IBM zSystems to have 8000 MB memory
3270port 3270                   # tn3270e connections specify this Linux port
processors 1                    # create one CP

[manager]
name awsckd 0008                # 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 ISV zPDT programs that emulate various device types. The number after the device manager name is an arbitrary hexadecimal number (up to 4 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 that is used for volume emulation, and various other parameters. The volume that is mounted at an address can be specified in the devmap or changed by running the **awsmount** command while ISV zPDT is running. In this example, the emulated tape volume in Linux file /z/SAINIT is already mounted when ISV zPDT is started. We can change the volume (while ISV zPDT is running) by running an **awsmount** command that specifies a different Linux file. (The files must be in the proper emulated format, of course.) This action corresponds to changing a tape volume on a tape drive or changing a disk “pack” in earlier years.

[20] However, the messages are added to a console log file. For more information about these log files, see 13.5, “zPDT log files” on page 282.

2.4.6 ISV zPDT directory structure in Linux

A Linux user running ISV zPDT has the following *default* directory structure in Linux:

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

Different Linux user IDs would have different default 1090 directories and files. The ISV zPDT operation is sensitive to the Linux user ID that is used. The usage of the default `logs`, `lists`, `srdis`, and `configs` directories is mandatory for some operations, but is optional for other files, such as emulated disk and tape volumes and for emulated card reader and printer devices. Emulated devices have default *file* names that are based on the assigned device number, but they can 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 disks and tapes subdirectories, and they are typically empty.)

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

Use a separate Linux file system for emulated disk volumes, which insulates them from Linux reinstallations and also insulates both the emulated volume file system and the base Linux file system (or systems) from unplanned growth in each other. For these reasons, most of the examples in this publication assume that all emulated I/O files are placed in the `/z` directory.[21] 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 directory to hold all the emulated volumes. The `cards`, `tapes`, `disks`, and `lists` directories, in the default directory path, are seldom used in typical operation.

There are no default file or directory names for ISV zPDT emulated volumes if you are not using locations such as `/home/<userid>/z1090/disks`. (Few zPDT customers use these default emulated locations.) In our examples, we tend to use file names that match the volser of an emulated disk volume, but this approach is not required.

2.4.7 PC LAN adapters

Both the base Linux and the IBM zSystems operating system can use (at the same time) a single Ethernet adapter (NIC) in the base PC. More than one adapter can be used, but this approach is usually unnecessary, and it can lead to external routing complications. Wired or wireless adapters can be used, although care must be taken not to disrupt a wireless connection when it is used for 3270 sessions or CTC connections.

For more information about LAN usage, see Chapter 7, “Local area networks” on page 149.

[21] The mount point name, `/z` in our examples, is arbitrary.

LAN setup is often the most complex part of ISV zPDT installation, especially when you are configuring both the base Linux LAN connections and the IBM zSystems operating system TCP/IP connections. Some degree of LAN and TCP/IP understanding is required for any ISV zPDT configuration beyond the most basic "local" 3270 operation.

2.4.8 ISV zPDT control structure

The general structure of the ISV zPDT control files is shown in Figure 2-2. This structure involves the ISV zPDT `awsstart` command, which has a parameter pointing to a devmap, and the devmap contains the names of the Linux files that are used for emulated devices (such as IBM 3390 volumes) and the amount of storage that is seen by the emulated IBM zSystems functions.

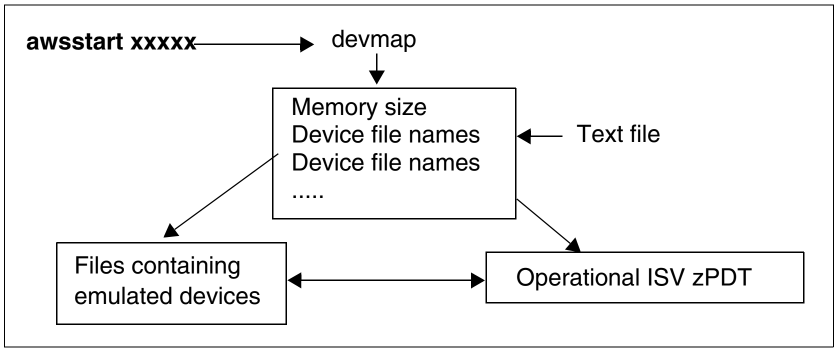


Figure 2-2 Control files: general structure

A devmap is a simple Linux text file containing specifications for the IBM zSystems machine.

2.4.9 ISV zPDT instances

Logging in to Linux and starting an ISV zPDT operation creates an *instance* of ISV zPDT usage. This instance might have one or more IBM zSystems CPs that are associated with it, depending on the licenses that are available and the parameters in the devmap. Then, if you log in to Linux with a second Linux user ID and start another ISV zPDT operation, this action creates a second *instance*. Multiple instances mean that multiple, independent ISV zPDT environments are run in parallel. The total number of CPs across all concurrent instances cannot exceed the number that is allowed by the token.[22]

A 1090 model L03 token can have up to three IBM zSystems CPs (or mixtures of CPs, zAAPs, and IFLs) plus up to three zIIPs. These CPs can be used for three ISV zPDT instances, each with a single CP, separate IBM zSystems memory,[23] and a separate IBM zSystems operating system. Alternatively, a single ISV zPDT instance could be used with one, two, or three CPs, which is more likely for most ISV zPDT users.

[22] Or by the total number of licenses from multiple tokens or software licenses, with a design limit of eight for any ISV zPDT instance.
[23] The combined IBM zSystems memory is subject to a later discussion about memory.

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 ISV zPDT instance must not be more than the number of processor cores on the base Linux system. If there are fewer cores than requested CPs, ISV zPDT does not start all the requested CPs.
- A full ISV zPDT operation can use more cores in the base PC system than there are IBM zSystems CPs that are defined in any one instance. The additional processors are used for I/O, help prepare IBM zSystems instructions for use, and for non ISV zPDT Linux processes.

In basic usage, emulated I/O devices are unique to an ISV zPDT instance. However, there are advanced ISV zPDT options that permit sharing emulated I/O devices among multiple instances. The minimum number of base processor cores, as stated earlier, should be *at least* equal to the maximum number of CPs in any instance. Otherwise, there is no association of particular base processor cores to CPs.

Most of this publication is focused on single instance operation. Chapter 10, “Multiple instances and guests” on page 215 provides setup and usage instructions for multiple ISV zPDT instances.

2.5 ISV zPDT releases

There have been many ISV zPDT releases over the years. The following sections summarize significant changes. In the tables that follow, the information about Linux levels is important. Lower-level Linux systems should not be used when running the associated ISV zPDT release. Other Linux distributions with libraries at an equivalent level (or later) might be used, although only the listed distributions were tried by the developers.

2.5.1 Current release

Key elements in GA11 (or in GA10 fix packs) include the following items, which are summarized in Table 2-3 on page 45:

- General implementation of IBM zSystems z16 instructions (with some exceptions), which involves extensive additions to the IBM zSystems system architecture.
- Corrections for IBM z/OS Container Extensions (zCX) startup problems.
- The Linux distribution levels that are used to build this release are updated from the ones that are used for the GA10 release.
- Fixes for obscure channel command word (CCW) program failures to detect end of file.
- Implementation of the DFLTCC instruction, but only in synchronous mode. However, the facility indication for this deflate function is turned off. The indicator can be turned on with the zPDT **dflt** command. For more information, see 4.2.31, “The dflt command” on page 93.
- Removes the need to run ISV zPDT with some access to Linux root_authority. However, root authority is still needed to install ISV zPDT or for some administrative commands. (This function was added in fix packs for the previous ISV zPDT release, but is consolidated in release GA11.)
- Multiple minor fixes for small problems, including a method of handling “dynamic sensing” when z/OS runs under z/VM.

- The MSA9 facility is included. This facility can be important for some of the more advanced cryptographic environments. The general Common Cryptographic Architecture (CCA) level is now 8.0.
- The zPDT **`ztrace`** command was added.
- An **`acptool`** command was added for specialized cryptographic control.
- Various minor fixes that are related to Ubuntu 20.04 were resolved in this release or in fix packs for the previous ISV zPDT release.
- OSA emulation changed from the OSAExpress5s level to the OSAExpress6s level.
- Minor problems with special usages of the CMPSC instruction were corrected.
- Cryptographic emulation is now level CEX8S. This change involved changing to a new format that zPDT uses to store the customer-provided master keys. If customer-encrypted data is carried forward from earlier zPDT releases, the customer must reinitialize the GA11 emulated cryptographic adapter master keys by using the same keys that were used on the previous zPDT system.
- The **`acptool`** command was added, although it is intended for limited usage.
- Some outdated commands (**`senderrdata`** and **`snapdump`**) were removed.
- An updated version of Coupling Facility Control Code (CFCC) is included.
- The zPDT emulation option for IBM zEnterprise Data Compression (zEDC) was removed.

Table 2-3 Version 1 Release 11. (known as GA11)

Characteristic	**Version 1 Release 11**
Date released	August 2022
Initial ISV zPDT driver level	z1090-1-11.57.06
IBM zSystems architecture level	z16
Linux levels that were used to build and test the ISV zPDT release. (The “official” levels.)	RHEL 8.6, SUSE Linux Enterprise Server 15 SP3, and Ubuntu 20.04
Informally tested Linux levels (Earlier levels are not recommended.)	CentOS Stream 8.6 and 9.0, Fedora 35 and 36, and Open SUSE 15.3
Tested z/OS levels	2.4 and 2.5
z/VM that were used during development	7.2 (not all functions)
z/VSE	No formal testing.
Tested Linux for IBM zSystems level	RHEL 8, with other testing in progress.
Machines that were used for testing	A wide range of servers and laptops
Virtual environments that were used during testing	KVM

2.6 Previous ISV zPDT releases

Key characteristics of earlier releases are provided in Table 2-4.

Table 2-4 GA 1 - 10

GA	Date	ISV zPDT	Architecture	Selected details
GA1	Oct. 2009	39.11	Z900 ALS3	► Base: RHEL 5.2 or openSUSE 10.3 ► Informal: Fedora or openSUSE 11 ► z/OS: 1.9 or 1.10 Note: Has a 32-bit version.
GA2	June 2011	41.21	z10 ALS3	► Base: RHEL5.3 or openSUSE 10.3 or 11.1 ► Informal: openSUSE 11.1 or Fedora ► z/OS: 1.10 or 1.11 Note: pdsUtil, listVtoc, Customized Offering Driver (COD), MSA 3 & 4, and a 32-bit version
GA3	March 2012	43.20	z196	► Base: RHEL 5.4, SUSE Linux Enterprise Server 11, or openSUSE 11.2 ► Informal: openSUSE 11.3 and 11.4, or Fedora 12 ► z/OS: 1.11, 1.12, or 1.13; z/VM 5.3, 5.4, or 6.1; or z/VSE 4.2, 4.3, or 5.1 Note: Remote license servers, devmap updates, CFCC17, LAN interface names, token RAS, 8 CP + specialty (32-bit version no longer available), shared direct access storage device, migration utility, and RDzUT
GA4, 4.1	Dec 2012	45.18	z196, EC12	► Base: RHEL 6.1 or openSUSE 11.3 ► Informal: openSUSE 11.3 or 12.2; SUSE Linux Enterprise Server 11 SP2; or Fedora 15 or 17 ► z/OS: 1.12 or 1.13; or z/VM: 6.1 or 6.2 Note: Many new instructions, zBX, virtualization, integrated consoles, z1090term, CEX4C, and CFCC18
GA5	Feb 2014	47.xx	EC12 GA	► Base: RHEL 6.x openSUSE 11.3 or SUSE Linux Enterprise Server 11 SP2 ► Informal: Fedora 17 or 19, or openSUSE 11.4, 12.1 ► z/OS: 1.12, 1.13, or 2.1; or z/VM: 6.2, and some 6.3 Note: CEX4, CP Assist for Cryptographic Functions (CPACF) updates, CFCC19, and SCSI 359x
GA6	March 2015	49.xx	z13	► Base: RHEL: 7.0 or SUSE Linux Enterprise Server 11 SP3 ► Informal: openSUSE 13.1, SLEX 11 SP3, or Fedora 20 ► z/OS: 1.13 or 2.1; z/VM 6,2 or 6.3; or z/VSE 5.1 or 5.2 Note: Many new instructions, CEX5S, CPACF updates, STP, CFCC20 SL16, r/o direct access storage device, KVM, and zBX
GA7	March 2017	49.xx	z13 GA2	► Base: RHEL 6.3 - 7.1, SUSE Linux Enterprise Server 11 SP2, or Ubuntu 16.04 ► Informal: Fedora 25, Leap42.1, or SUSE Linux Enterprise Server 11 SP3 ► z/OS: 1.13, 2.1, or 2.2; z/VM 6.2 - 6.4; or z/VSE 6.1 Note: Ubuntu base, z/TPF, software license server, CFCC21, up to 2048 devices, zBX dropped

GA	Date	ISV zPDT	Architecture	Selected details
GA8	Dec 2017	51.xx	IBM z14	► Base: RHEL 7.3, SUSE Linux Enterprise Server 12 SP1, Ubuntu 16.04, or Leap 42.1 ► Informal: Leap 42.1, openSUSE 13.1, or Fedora 25 ► z/OS: 2.1, 2.2, or 2.3; z/VM 6.2 - 6.4; or z/VSE 6.1 Note: New instructions IBM z14, “free” zIIPs, CFCC22, revised awsckd (2017, MIDAW), jumbo frames, and I/O counts by using awsstat
GA9	March 2019	53.xx	IBM z14 GA2	► Base: RHEL 7.5, SUSE Linux Enterprise Server SP3, or Ubuntu 18.04 ► Informal: Leap 42.3; Leap 15; Fedora 27 or 28; or CentOS 7.0 or 7.4 ► z/OS: 2.2, 2.3, or 2.4 Note: Special tape CCWs, LPAR name, console file, CFCC23, zEDC, channel measurements, layer 2 fixed, and 64-bit support for gen2 licenses
GA10	March 2020	55.xx	z15	► Base: RHEL 7.7, SUSE Linux Enterprise Server 12 SP3, or Ubuntu 16.04.6 LTS ► Informal: RHEL 7.x, 8.0; SUSE Linux Enterprise Server 12 SP3; openSUSE Leap 42.3 or 15.0; or Ubuntu 16.04LTS or 18.04 ► z/OS 2.2, 2.3, or 2.4; or z/VM 6.4 or 7.1 Notes: DFLTCC missing, CFCC level 24, Crypto 7S, MSA9 missing, and various bugs fixed.

3

Device maps for ISV IBM Z Program Development Tool

In this chapter, we provide reference information for Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) device map (devmap) entries for the IBM zSystems processors and I/O device managers. Information and guidance for using these device managers are found throughout this publication. Devmaps are generally not included with ISV zPDT distribution packages. However, a devmap must be created before ISV zPDT can be started.

This chapter describes all devmap options (including all the supported device managers) in some detail. As a practical matter, most ISV zPDT operations involve only a reasonable subset of the possible options. When reading this material for the first time, do not spend too much energy studying options that you are unlikely to use.

3.1 Device maps

A *devmap* consists of a system stanza, optional adjunct processors (APs), IBM zEnterprise Data Compression (zEDC) (if appropriate), Server Time Protocol (STP) stanzas, 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 use. Usage information is provided in other chapters of this publication.

A devmap is a simple Linux text file with an arbitrary file name. Many devmaps might exist, but only one can be in use for an instance of ISV zPDT. It is possible to have multiple ISV zPDT instances running, each under a different Linux user ID. Each instance has its own devmap. The remainder of this chapter assumes that a single instance of ISV zPDT is used.

Create a devmap file in lowercase,[1] except for parameters that specify Linux file names. Devmap parameters begin in the first column of each statement. Stanzas are separated by blank lines. A number sign (#) signals the beginning of comments in a line. The square brackets that shown in the descriptions below are part of the syntax and must be entered as shown.

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

3.2 System stanza

A [system] stanza might look like the following example:

```
[system]
memory 16G                # 16 GB memory for IBM zSystems -- Use G and GB
processors 1              # number of CPs
3270port 3270             # specify unique IP port number for aws3274
expand 0m                  # no expanded storage. mostly obsolete
ipl 0A80 “0A8200”         # automatic ipl control (optional; not recommended)
cpuopt alr=on             # optional function (defaults to on)
command 2 x3270 localhost:3270     #Linux command run through the devmap
command 2 x3270 -model 3279-5 localhost:3270   #various x3270 options available
```

The “square brackets” around the system keyword are required. They also appear in other devmap elements and are required as shown in the text in this publication.

The **memory** statement specifies the size of the IBM zSystems memory to be emulated for an ISV zPDT operation. The number that is specified must be smaller than the maximum shared memory value that is specified for Linux, which must be set by the **kernel.shmmax** parameter in Linux.[2] For z/OS, the memory value should normally be at least 4G. We typically use 8G - 12G, but the value might be much larger.[3]

[1] Lowercase is not required by some elements of a devmap, which ignore uppercase or lowercase. Not all devmap elements ignore case. To avoid problems, use lowercase for everything (except for Linux file names, which are case-sensitive).

[2] This kernel variable is specified by the instructions in Chapter 5, “ISV IBM Z Program Development Tool installation” on page 125.

[3] The memory size can also be specified in megabytes by using a “M” suffix.

You must have a **processors** statement unless your devmap is for a group controller. (For more information about group controllers, see Chapter 10, “Multiple instances and guests” on page 215.) The **processors** statement specifies the number of IBM zSystems processors to be used in this instance. The default is 1. The **processors** statement is also used to indicate the usage of specialty processing units (PUs) as in the following examples (where `cp` indicates a normal, general-purpose CP):

```
processors 3                # three CPs. Assumed “cp” type. 3 licenses needed.
processors 3 cp cp ziip     # two CPs and one zIIP; 2 licenses needed
processors 2 cp cp ziip     # invalid. Two processors, but three definitions
processors 1 ziip           # invalid. Must have at least one cp or ifl
processors 3 cp ziip ifl    # one of each; needs 2 licenses
processors 3 ziip cp cp     # invalid; cp must be first in the list
processors 4 cp cp cp ziip  # only 3 licenses needed; ziip is “free”
```

The operands for the **processors** statement are the number of processors (typically 1, 2, or 3)[4], which (excluding IBM zSystems Integrated Information Processor (zIIP) processors) cannot exceed the number that is allowed by the number of licenses that is allowed by the ISV zPDT token. The processors default to CPs; if you want specialty processors, list them after the number, as shown in the examples. The processor types are `cp`, `ziip`, `zaap`, and `ifl`. If zIIPs or IBM zSeries Application Assist Processor (zAAP) processors are specified in the processors statement, at least one CP must be listed first.[5] Specialty SAP processors are not used and Coupling Facilities (CFs) are not specified in the devmap.[6]

The zAAP processors might not be useful in the IBM z13 (or later) environment that is created by ISV zPDT GA6 or later. Starting with ISV zPDT GA8, zIIP processors do not require an ISV zPDT license, although they are included in the maximum of eight processors for an ISV zPDT instance and are included when determining the minimum number of personal computer (PC) cores that are needed. At the time of writing, the number of zIIPs cannot be greater than the number of CPs.

The **expand** statement specifies the size of expanded storage for the IBM zSystems machine. This statement is optional. z/OS no longer uses expanded storage. However, some older releases of z/VM still might use it. Do not specify this option unless you have a specific need for it.

The **3270port** statement specifies a port number to be used by the base Linux TCP/IP for the aws3274 device manager. This port must be an unused one on the base Linux and is typically a number greater than 1024. In this publication, we arbitrarily use port 3270 because it is easy to remember. A TN3270e emulator connection to this Linux port appears as a local, channel-attached 3270 to the IBM zSystems processors.

The **ipl** statement is optional and indicates that the **ipl** command is to be run automatically when the ISV zPDT operation is started. However, using this option might prevent you from connecting 3270 emulator sessions in a timely way. Use this option carefully, if at all.

A **cpuopt zVM_CouplingFacility** statement (not shown in the above example) should not be used with ISV zPDT. In effect, the **zVM_CouplingFacility** function is always present for ISV zPDT systems. Other **cpuopt** parameters produce non-standard configurations. They are described in 13.3, “The cpuopt statement” on page 279.

[4] The number of processors for an instance has a maximum value of 8. You might need multiple ISV zPDT tokens when individual tokens have a maximum of three licenses.

[5] The first processor type that is listed becomes the initial program load (IPL) processor. zIIPs and zAAPs cannot handle an IPL.

[6] ISV zPDT uses z/VM to provide emulated CFs.

The **command** statement specifies Linux commands that are automatically run as part of ISV zPDT operation. The syntax is as follows:

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

The phase number is a digit 1 - 4:

- Phase 1 means that the command is run before ISV zPDT starts.
- Phase 2 means that the command is run after ISV zPDT is initialized.
- Phase 3 means that the command is run before ISV zPDT is shut down.
- Phase 4 means that the command is run after ISV zPDT is shut down.

By default, commands are run asynchronously but can be forced to run synchronously. (This approach should seldom be used because it forces other ISV zPDT operations to wait until the command completes. For example, do not use the synchronous approach for an x3270 operation.) The word command can be abbreviated to cmd, and synchronous can be abbreviated to sync. If asynchronous commands terminate while ISV zPDT is still running, they are not automatically restarted. If the commands are still running when ISV zPDT is shut down, they are sent a SIGTERM signal and should terminate.

Additional system stanza options include the following ones:

```
[system]
...
rdtserver 27000@our.server.acme.com    # not used with ISV zPDT
int3270port 3271                       # HMC-style 3270 integrated port
intASCIIport 3300                      # HMC-style ASCII port
system_name mybase                     # specify the “LPAR name”
oprmsg_file /home/ibmsys1/opmessages   # for “HMC hardware console” output
```

The **int3270port** and **intASCIIport** statements provide emulation for Hardware Management Console (HMC)-style integrated terminal functions. The operand for each statement is a port number. After starting ISV zPDT with one of these operands, you start a 3270 emulator that is connected to the indicated Linux port number or start **z1090term**[7] connected to the indicated ASCII terminal port number. These emulated terminals do not need to be on the base Linux system.

The 3270 terminal session that is associated with the int3270port function should[8] be a 3270 model 3 (with a 32x80 screen). Any other 3270 terminal model *might not* connect properly. We found that, in some cases, the smpppd rpm must be included in Linux for the connection to work. In general, the int3270port function is used only when installing z/VM from the formal IBM distribution media. It is not needed when using an Application Development Controlled Distribution (ADCD) z/VM distribution.

[7] For more information, see the command descriptions in Chapter 4, “ISV IBM Z Program Development Tool commands” on page 71.

[8] This requirement appears to vary with z/OS releases. If you have a problem using the int3270 function, be certain you are using the correct 3270 model type.

Tip: Our examples involving int3270port use 3271 as the port number. There is nothing special about this port number; any unused port number can be selected. We found that when attempting to connect a local x3270 session to int3270port the usual link of "localhost:3271" sometimes did not work, while "127.0.0.1:3271" did work.

Restriction: The int3270port and intASCIIport functions have limited applicability. They should not be included in a devmap unless there is a specific need for one or the other one. Their usage can interfere with the default output of "HMC console" messages to a Linux window and the zPDT **`oprmsg`** command. We also found that in some environments including both options in the same devmap created issues.

By default, ISV zPDT uses the Linux user ID that was used to start ISV zPDT as the instance name (the "LPAR" name or "CPCNAME").[9] An LPAR name cannot be longer than 8 characters, which restricts the Linux user ID to no more than 8 characters. However, the **`system_name`** option in the devmap (maximum of 8 characters) can be used to specify the "LPAR" name, removing the length restriction on the Linux user ID that is used to start ISV zPDT. The specified name is automatically converted to uppercase Extended Binary Coded Decimal Interchange Code (EBCDIC).

By default, ISV zPDT sends output that is directed to the "HMC hardware console" to the Linux command-line interface (CLI) that was used to start ISV zPDT. (If this CLI is closed, the output is not displayed.) By default, this output (along with any **`oprmsg`** commands) is written to an ISV zPDT log file. The **`oprmsg_file`** option specifies a Linux file that is to receive such messages.[10] If an **`oprmsg_file`** is specified, the messages are no longer written to the default ISV zPDT log file. The file name is case-sensitive. This option has several implications:

- The messages are available in the specified Linux file whether the original Linux CLI is open or closed.
- If the **`oprmsg_file`** option is used, the default ISV zPDT log file will not contain the messages that are written to the specified **`oprmsg_file`**.
- Typically, this option is used with automation (running under Linux) that generates **`oprmsg`** commands that are sent to the emulated IBM zSystems. The responses, along with other operating system output, are placed in the specified Linux file where they can be inspected by the automation programs.[11]

The specified **`oprmsg_file`** can be deleted at any time by the user (working with Linux commands or programs). It will be re-created (empty) when the next message arrives for it.

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

```
[system]
memory 8G
processors 2
3270port 3270
command 2 x3270 -model 4  localhost:3270
command 2 x3270 -model 4  localhost:3270
command 4 echo 'ISV zPDT operation has completed'
```

An ampersand (&) is not used after the x3270 commands in a `[system]` stanza. Also, the x3270 sessions are automatically closed when ISV zPDT is shut down.

[9] ISV zPDT does not generally support logical partition (LPAR) functions, but the LPAR term is often used to describe an ISV zPDT instance.
[10] This function is described in more detail in 13.5, "zPDT log files" on page 282.
[11] ISV zPDT does not provide any such automation functions or examples.

A devmap has several additional features,[12] as shown in the following list:

- 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 the following stanza:

```
[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 parentheses, 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=6500m
```

This command can be issued before an **awsstart** command (in the same Linux terminal window), but then it will not be effective in other Linux terminal windows. Alternatively, the **export** command can be added to the `.bashrc` file, where it is effective for any terminal window that is then opened. For practical purposes, we suggest adding any devmap environmental variables to the `.bashrc` file.[13] If the specified environmental variable is not set, a null string is placed in the devmap.

The **include** function in the example operates as you might expect. The file that is specified is logically inserted into the devmap at the point that is 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 that is specified cannot contain blanks. The name can be an environmental variable instead of a file name, for example, **include $(fileVAR)**. If the specified environmental variable is not defined, the **include** function is skipped.

[12] These features were added for special purposes. We have not seen much usage by typical ISV zPDT users.

[13] Other required changes to the `.bashrc` file are described in Chapter 5, "ISV IBM Z Program Development Tool installation" on page 125.

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

```
[system]
memory 6000m
processors 3 cp cp ziip
3270port 3270
int3270port 3271
intASCIIport 4000
rdtserver 6700@192.168.1.220
expand 1000m                  #Who uses expanded memory today?
system_name old22             #maximum more than 8 characters
oprmsg_file /tmp/bill/messages
#ipl 0A80 "0A8200"            (not recommended. Commented out here)
cpuopt alr=on
message This devmap is excessive
command 2 x3270 localhost:3270
command 2 x3270 localhost:3270
command 2 x3270 localhost:3271
message Remember to start more 3270 sessions
include devmap2
```

As a practical matter, typical devmaps use only the **memory**, **processors**, **3270port**, and possibly a few **command** statements. The other statements tend to be for special cases.

3.3 Adjunct-processor stanza

The ISV zPDT system optionally provides emulation of the IBM zSystems cryptographic adapter.[14] The release level of the cryptographic adapter varies with the ISV zPDT release level. The basic devmap format for this emulation is as follows:

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

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

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

This stanza indicates that the instance is using domain 2 in cryptographic coprocessors 0 and 1. For more information, see Chapter 17, "Cryptographic usage" on page 331. The second **domain** number option should not be used for a single-instance ISV zPDT configuration.

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

3.4 Server Time Protocol stanza

The Linux daemons that are associated with STP must be started before you use a devmap that contains an STP stanza. The stanza is as follows:

```
[stp]
ctn 00000000F1F0F9F0       #16 hex digits beginning with 00
node 1 W520 *              #asterisks marks this node
node 2 W510
```

All ISV zPDT systems participating in a CCT/STP network must have a similar stanza (with the asterisk denoting the local Linux name). When a devmap includes an `stp` stanza, the devmap cannot be started (with an **`awsstart`** command) unless the STP function is active on the base Linux system. The devmap stanza can also include a **`LEAPSECONDS`** statement, which is not shown in this example. Few ISV zPDT users use this function.

3.5 zEDC stanza

zPDT GA9 provides limited zEDC emulation. The required stanza can look like the following one:

```
[pcipchid]
name awszedc 120 zedc
function 0c0 3
```

This example has `pchid=120`, zEDC address 0c0, and virtual function 3. Operands such as these are required for zEDC operation, but the operand values are not relevant to ISV zPDT use. The IBM zSystems DFLTCC instruction, which is present in ISV zPDT GA11, replaces zEDC usage, and zEDC devmap statements are ignored with GA11.

3.6 Device manager stanzas

A device manager stanza has the following general format:

```
[manager]
name awsckd C700
device 0a80 3390 3990 /z/SBRES1
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 the name can be any of the supported device managers. The device manager name is followed by an arbitrary hex number (up to 4 digits, which is different for each name statement).[15] The **`name`** statement is followed by as many **`device`** statements as needed.

[15] 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.

The general format is as follows:

- For name statements:
  - The constant “name” starting in the first column.
  - The device manager name, such as awsckd.
  - A hex number. Each **`name`** statement must have a different number.
  - More optional parameters, such as:
    - **`--path=xx`** to specify an emulated CHPID number.
    - **`--pathtype=xxx`** to specify an emulated CHPID type (usually emulated I/O (EIO)).
    - **`--compress`** to specify compressed awstape generation.
    - Various optional **`tunnel`** parameters for an Open Systems Adapter (OSA) operation.
- For device statements:
  - The constant “device” starting in the first column.
  - The device number (“address”) to be used, which is expressed in hexadecimal. This number can be three or four digits.
  - The IBM device type, such as 3390. It must specify a correct device type for the device manager.
  - The IBM control unit type that is associated with the device, which can be up to 4 characters. (This parameter is not used for anything at the time of writing, but a wise approach is to use an appropriate control unit number.)
  - Parameter (or parameters) unique to the device:
    - A fully qualified file name.
    - **`--unitadd=x`** to specify a unit address 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.
    - More 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[16] for most device managers. In rare cases, you might change these values by using the **`--path`** and **`--pathtype`** operands on a name statement, as follows:

```
[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 can be used. A maximum of 256 devices can be listed in a stanza. The device numbers (addresses) that are assigned to each device do not need to be sequential or in any particular order.

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

3.6.1 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 [--shared]
device a80 3390 3990 /z/SYSRES
device a85 3390 3990 /tmp/my3390vol
device 0aa7 3390 2107 /z/SARES1
device 0AA8 3390 3990            #No file specified; can use awsmount
etc
```

The device type can be 3390 or 3380; in either case, the Linux file that is named by the fourth parameter of the device statement 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.

Tip: Do not select 3380 device types unless you have a particular need for them. This device type is rarely used today.

Also, Linux file names containing space characters are not supported.

The third operand of a device statement is a control unit type. This information is not used by ISV zPDT, but it is a positional operand that must be specified. Starting with ISV zPDT GA 8, awsckd emulates IBM 2107 control units. However, you can specify 3990, 2107, or anything else up to 4 characters.

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 64 K-1 cylinders. ISV zPDT also supports extended address volume (EAV) 3390s. The volume size is specified when using the **alcckd** command to create the volume.

The *extra* cylinders of a 3390 are not emulated; they are the cylinders that are reserved as spares or for diagnostic use. For example, a 3390-3 contains 3339 usable cylinders, and these cylinders are what is emulated. Parallel Access Volumes (PAVs) are not supported.

Device statements can omit a file name. In this case, the indicated unit (3390 at address 0AA8 in the example) exists, but has no volume that is mounted. A volume (which is a Linux file in the appropriate CKD format) can be mounted (or dismounted) while ISV zPDT is operational, which provides dynamic changes to the direct access storage device environment without stopping ISV zPDT. The **awsmount** command is used for this operation.

The **--shared** option is relevant only if the volume (that is, the Linux file) is shared among multiple z/OS systems. This option causes the awsckd device manager to perform the following actions:

1. Emulate **RESERVE** and **RELEASE** channel commands.
2. Lock (at the Linux level) the logical tracks of the `ckd` volume while they are addressed by an active IBM zSystems channel program.

Much more is involved in sharing IBM zSystems volumes, and the **`--shared`** option is only one element that is involved. Use this option when multiple ISV zPDT instances (in the same Linux) are sharing direct access storage device volumes and when separate ISV zPDT systems (on separate Linux bases) are sharing direct access storage device through a Linux shared file system. Proper serialization, as seen by z/OS, is essential for shared direct access storage device, and implementing such serialization typically involves z/OS global resource serialization (GRS).[17] The **`--shared`** option is *not* used when sharing direct access storage device volumes among multiple z/VM guests.

3.6.2 The awsfba device manager

The awsfba device manager provides emulation for Fixed Block Architecture (FBA) disk devices[18] (as used by z/VM and a few other operating systems), as shown in the following stanza:

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

The awsfba devices (volumes) must be created before they can be used. This task is accomplished by using the **`alcfba`** utility. This device manager does *not* support the more recent fiber open system FBA devices for z/OS.

ISV zPDT development performs only minimal testing for these FBA devices. As a best practice, use CKD disks unless there is a specific need for FBA disks.

3.6.3 The aws3274 device manager

The aws3274 device manager emulates local, channel-attached, DFT, non-Systems Network Architecture (SNA) 3270 sessions. These sessions are used for MVS operator consoles, simple IBM Virtual Telecommunications Access Method (IBM VTAM®) sessions (Time Sharing Option (TSO), IBM CICS, and so on), z/VM terminals, and similar purposes. The actual 3270 emulators (x3270, IBM Personal Communications, or other 3270 emulators) might be local (on the underlying Linux system running ISV zPDT) or remotely connected through 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 IBM zSystems software. The same physical Ethernet interface can be used for Linux functions, such as Telnet, SSH, aws3274, and FTP, and also for emulated OSA connections.

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

The devmap parameters for emulated local 3270s offer several options, which are best explained by an example:

```
[manager]
name aws3274  C700                # C700 is an arbitrary CUNUMBR
device 0700 3279 3274
device 0701 3279 3274 L701
device 0702 3279 3274 L702
device 0703 3279 3274 TSO
```

[17] GRS is a basic element of an IBM zSystems sysplex configuration.
[18] These devices had IBM type numbers, such as 3370, 9332, 9335, and 9336.

```
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
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 optional operand controls potential 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 LU name, intended for SNA protocols, during startup. We use this LU name passing facility here without passing it to VTAM.) The LUnames can have a maximum of 11 characters.

In this example, LUnames are L701, L702, TSO, and IMS. The connection rules are as follows:

- The LUname is not case-sensitive.
- If an LUname is specified by the TN3270e client, then a free device with the matching LUname is used.
- If no LUname is specified by the TN3270E client, the next free device in the list is used, regardless of whether it has an associated LUname.
- 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.

The aws3274 device manager listens on a port in the base Linux TCP/IP system. Assume that the Linux TCP/IP address is `192.168.1.80` in the following examples. Also, assume that our devmap specifies 3270 as the aws3270 port number. A user can enter one of the following Linux commands to establish an x3270 session:

```
$ x3270 -port 3270 192.168.1.80 &            case one
$ x3270 -port 3270 TSO@192.168.1.80 &        case two
$ x3270 -port 3270 L702@192.168.1.80 &       case three
$ x3270 -port 3270 IMS@localhost &           use local system
```

Assume that our x3270 client is on a remote machine that is connected to a private local area network (LAN) that includes the ISV 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 if that device is free currently. The fourth example illustrates that the same LUname rules apply to connections from the Linux desktop. You might use a shorter form of the **x3270** command, as follows:

```
x3270 localhost:3270 &
```

In the examples, 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 ADCD z/OS system might be defined as follows, which is the most common example for ISV zPDT users:[19]

```
[manager]
name aws3274 C700
device 0700 3279 3274
device 0701 3279 3274
device 0702 3279 3274
device 0703 3179 3274
.....
device 070A 3179 3274
```

Client 3270 connections take the next free terminal in the devmap list if no LU conditions are specified. This action can be especially useful if the first terminal in the devmap is the MVS operator console[20] and the next terminal is a suitable TSO address. In this case, without specifying any LU names, the first x3270 session is the MVS operator console, and the second session will be a TSO session (or CICS or some other VTAM application). As a practical matter, we observed that few ISV zPDT customers use the LU name facility.

From the user's perspective, a 3270 terminal is a TN3270e session. The IBM Personal Communications product and the x3270 emulator that are available for Linux were tested for this usage.[21] The TN3270e client might operate on the machine running the ISV zPDT processes (on the local Linux desktop, 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 IBM zSystems operating system.

The usage of TN3270e (rather than 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.

Starting a 3270 session (through aws3274) requires a small amount of free space in the Linux `/tmp` file system. If `/tmp` is full, a new aws3274 session cannot be started.

3.6.4 The awstape device manager

The awstape device manager provides "emulated" tape drives that use Linux files as the "tapes" that are involved. This device manager is not related to "real" tape drives that might be attached to the Linux system. Definitions for awstape appear as follows:

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

[19] You might notice that the third operand in the aws3274 examples is sometimes 3174 and sometimes 3274. This operand in device statements is not processed, and it can be any 4-character value. We suggest a meaningful value only for documentation.

[20] The ADCD z/OS systems define the MVS console at address 700.

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

The emulated device type can be 3420, 3480, 3490, or 3590. (The third operand, the control unit type, is not meaningful.) A file name can 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 like a premounted tape on a larger IBM zSystems server. 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 number of bytes are written. (Then, z/OS probably writes trailer labels and calls for another tape mount.) If **`maxlength`** is not specified, then the maximum tape content is limited by one or more of the following conditions:

- The amount of free disk space in the Linux file system.
- An architectural limit of approximately 4 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.

The **`--MNTCCWS`** parameter enables use of 4B and 54 channel command words (CCWs), which are used to manipulate the Linux file name that is associated with the emulated tape drive. This process is described in 13.19.5, “Using special channel command words to manipulate tape volume” on page 297.

Emulated tape volumes that are created through this device manager are in awstape format, and they can be exchanged with other systems that can process this format. All awstape files are compatible with all ISV zPDT emulated tape devices. An awstape file that is written by an emulated 3490 can be read by an emulated 3420, for example.

The correct responses for hardware compaction (IDRC) are emulated, although tape data is not compacted by this method. The awstape data can be optionally compacted by the awstape device manager, which is controlled through a devmap or an awsmount parameter. The compaction format is unique to ISV 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, that is, it is not necessary to create or initialize the volume before writing to it unless the software expects a labeled tape. (An “empty” labeled tape can be created with the ISV zPDT **`aws_tapeInit`** command.)

Tip: On some IBM zSystems servers, there are various models of 3590 tape drives that are available that can present various and complex information about various “sense” commands. The zPDT emulated 3590 does not provide all of this information, which in rare cases might create a software problem. If this situation happens, emulate one of the other tape drive types that are available. (The only specific instance we know about involves DFSMShsm creating a new stored data set.)

3.6.5 The awsosa device manager

The awsosa device manager emulates various OSA-Express functions that are used by IBM zSystems TCP/IP or SNA.[22] At the time of writing, the emulation level is OSA-Express6s. Two manager formats are available:

```
[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
device 405 osa osa
device 406 osa osa
```

The first example is used with a typical PC Ethernet adapter. The second example is for a *tunnel interface* between the emulated OSA adapter and the underlying Linux TCP/IP system.[23] 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 with each other through it. Both OSA and Linux can share the adapter for connection to external TCP/IP systems, but they cannot communicate with each other.[24] A tunnel interface (which is like another Ethernet adapter) is necessary for direct communication between the underlying Linux system and the IBM zSystems OSA operation.

Tip: For more information about using the emulated OSA functions, see Chapter 7, “Local area networks” on page 149.

The **`--path`** operand specifies a CHPID number. The correct number is determined with the **`find_io`** command. For these examples, we assume that the CHPID for wired Ethernet is F0 and the CHPID for a tunnel interface is A0. The **`--pathtype`** is OSA-Express Direct (OSD) (for QDIO) or OSA-Express (OSE)[25] (for LAN Channel Station (LCS) or non-QDIO). In some cases, the **`find_io`** command does not provide a CHPID (path name) for a LAN interface, so the **`--interface=xxxx`** parameter can be used to name a specific LAN interface. The interaction of the **`--path`** and **`--interface`** parameters is explained in detail in Chapter 7, “Local area networks” on page 149. Here is an example of using the **`--interface`** parameter:

```
name awsosa 1234 --path=B0 --pathtype=OSD --interface=em1
```

The **`--unitadd`** operands specify the internal OSA interface number, which normally is not needed for QDIO operation. They might 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.

The z/OS device type should be OSA, as shown in the z/OS input/output definition file (IODF) (and when displaying devices on the MVS console).[26]

[22] ISV zPDT SNA usage has not been tested by IBM, and there is no support for it.
[23] If no IP address is specified for a tunnel interface, it defaults to 10.1.1.1.
[24] This odd limitation is a characteristic of current Linux implementations.
[25] We do not recommend that you use OSE unless you have a special need for it.
[26] Older z/OS systems can use channel-to-channel (CTC) device definitions for these interfaces, especially when they are used for TCP/IP. These definitions should be replaced with device type OSA.

The `--pathtype` parameter must specify OSD or OSE. When used in OSE mode, the OSA interfaces are associated with OSA Address Table (OAT) definitions that specify how each interface is used. At the time of writing, Open Systems Adapter/Support Facility (OSA/SF) (the z/OS program to manage OATs) is not available with z/OS releases, and its partial replacement, the `QUERYINFO` command, is *not* supported by ISV zPDT. SNA usage requires CHPID type OSE, although there is no ISV zPDT support for SNA usage. Do *not* use OSE mode unless you have a specific requirement for it.

Table 3-1 provides details about OSA-Express emulation at the time of writing.

Important: The larger specifications here should not be taken as reasonable designs for general ISV zPDT usage. These numbers relate to fixed table sizes in a zPDT OSA implementation. Actual usage of larger configurations (within the limitations that are shown in this table) might involve other considerations.

Table 3-1 OSA-Express limits, per port

Item	Value
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 address stacks per port (OSD or OSE)	16
SNA PUs per OSA-Express port (SNA is not supported for ISV zPDT.)	512
OSE subchannels per stack	2
OSE or OSD maximum devices	48
OSE IP address stacks per OSA port or CHPID	16
OSD subchannels per stack	3
OSD subchannels per OSA or CHPID	48
Is OSA/SF or QUERYINFO available?	No
Is Generic VLAN Registration Protocol (GVRP) supported?	No

3.6.6 The awsrdr device manager

The awsrdr device manager emulates an IBM 2540 card reader. Only one awsrdr device can be configured for an instance of an ISV zPDT operation. Typically, the emulated card reader is used to submit jobs to the operating system.[27] If we assume that the operating system is z/OS, then Job Entry Subsystem (JES) 2 or JES3 should be configured with a "hot" reader.[28] The traditional address for an IBM 2540 is 00C, which we use in our examples.

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

[28] The term "hot reader" means that there is always a read outstanding for the card reader. When an operator places cards in the reader, JES begins reading them.

The awsrdr device manager monitors the directory that is specified in the device statement. When a file is found in the directory, it is read (assuming an IBM zSystems program has a *read* that is outstanding for the card reader, which is the case with a JES hot reader). After the file ("card deck") is read, it is moved to the *old* subdirectory. So, there is never a file in the directory that is assigned to the reader other than a file that you moved there to be read. When file is read, it is moved out of the reader directory. If awsrdr is not active or if there is no IBM zSystems program trying to read cards, then files sit in the reader directory indefinitely.

The devmap entry for the card reader might look like this:

```
[manager]
name awsrdr 010C
device 00C 2540 2821 /home/ibmsys1/cards/*    (the asterisk is required)
```

The `/home/ibmsys1/cards/` directory in the example is arbitrary. The default path is `/home/<userid>/z1090/cards/`.

ASCII and EBCDIC

Linux text files are normally in ASCII. z/OS cards are normally in EBCDIC, but they might contain binary or other formats for information. A "real" card reader uses fixed-length records (80 bytes), but a Linux text file has variable length records that terminate with an New Line (NL) character.

The conversion rules are as follows:

- If the input file name (in the directory that is used by awsrdr) contains the suffix `.ebc` or `.bin`, then the file is assumed to be in EBCDIC format 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 converted to EBCDIC when it is read.
- If the input file contains the ASCII characters `//`, `ID`, `$$`, or `USERID` in the first bytes, the file is assumed to be in ASCII and converted to EBCDIC.
- If none of these conditions are true, then the file is assumed to be EBCDIC (or binary as used for IBM zSystems) and 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 these reasons, then awsrdr reads it in 80-byte chunks and passes the data (unchanged) to the emulated card reader.

Another way to convert ASCII text files to EBCDIC card files is by running the **`txt2card`** command.

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

3.6.7 The awsprt device manager

The awsprt device manager emulates an IBM 1403 or 3211 printer. Forms Control Buffer (FCB) functions[29] are supported for 3211 emulation, but Universal Character Set (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 them.

[29] As a best practice, do not use FCB functions because they have not been tested recently.

awsprt cannot recognize divisions between IBM zSystems jobs. It just concatenates all output (potentially from multiple jobs) into the output file. The device statement specifies the output file to use:

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

If a file name is not provided with the device statement, 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 or LF characters instead of NL characters.

The **`awsmount`** command can be used to close the existing output file and open a new output file. The previous output file is closed, and then it is available for display or printing under Linux. For more information about printing output with the awsprt device manager, see 12.10, “Local printing” on page 259.

3.6.8 The awscmd device manager

This device manager provides a device that appears to IBM zSystems software as a tape drive. Its function is to send a command (and data) to the underlying Linux and then receive the output from the Linux command. Any Linux command might be sent, including the ones that might destroy the Linux system.[30] Obviously, this device manager should be used with care, and it might not be appropriate for an ISV zPDT environment that can be accessed by untrusted users.

Configuration is like other device managers:

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

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

The intended operation (by an IBM zSystems application program) is as follows:

1. A rewind is issued to the device.
2. The 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. The awscmd device manager submits the command (and data) to Linux through a shell that does not appear on the Linux panel. The current Linux directory for the command is the same directory that was used to start ISV 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

[30] The Linux commands run with the authority of the user ID that started ISV zPDT operation.

  - The stderr data from the Linux command
  - The return code (converted to characters) from the Linux command
8. The output (on the pseudo tape) was converted to EBCDIC.
9. Two tape marks are at the end of the pseudo-tape.

Restrictions

The command that is sent to Linux cannot include any redirection (less than (<) or greater than (>) characters), asynchronous indicator (ampersand (&) character), or pipe ("|" or vertical bar character). The pseudo-tape device appears as busy while Linux is running the command. Any Linux command that creates substantial delays (of many seconds) might cause I/O timeout errors to be generated in z/OS.[31]

At the time of writing, the following characters did not survive the conversion from EBCDIC to ASCII when included in the stdin data:

- Tilde (~)
- Caret (^)
- Colon (`:`)
- Double quotation marks (`"`)
- Less than (<)
- Greater than (>)
- Question mark (?)

An extended example of awscmd usage is described in Chapter 11, "The awscmd device manager" on page 227.

3.6.9 The awsscsi device manager

The awsscsi device manager emulates a mainframe tape drive by using a physical SCSI tape drive. The specific adapters and tape drives that are supported are described in Chapter 14, "Tape drives and tapes" on page 303. The general format is as follows:

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

The last operand of the device statement denotes the SCSI device to be used, which must be given as a `/dev/sgx` name and not a `/dev/stx` name. The differences are complex. Chapter 14, "Tape drives and tapes" on page 303 describes methods for determining the correct `/dev/sgx` name. The SCSI tape drive appears as a 3420, 3480, 3490, or 3592 device to the IBM zSystems software.[32] The **`awsmount`** command can be used with SCSI tape devices.

3.6.10 The aws3215 device manager

The aws3215 device manager provides emulation of a 3215 console by using devmap parameters like the following ones:

```
[manager]
name aws3215 AC00
```

[31] Earlier versions of awscmd had a timeout function that limited the time that was allowed for the Linux command. This timeout check was removed at customer requests.

[32] BM 3490 tape units have their own characteristics, one of which is a maximum block count of approximately 4 million (a 22-bit number).

```
device 009 3215 3215
```

It is possible but unusual to have multiple 3215 devices. Thee input to the 3215 console is done by running the **awsin** command in the Linux CLI. The output appears in the Linux panel that is used for the **awsstart** command.

3.6.11 The awsoma device manager

The awsoma device manager is used to read CDs or DVDs[33] that are written in a special format that is known as OMA. OMA is for input only because it is not possible to write to an awsoma device. In earlier days, z/VM and z/VSE were available in OMA format, and some Linux distributions for IBM zSystems might 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 specific file name. The second name is *relative* to the path that is specified by the first name.[34]

In a Linux based ISV zPDT system, the net effect is that the two names are concatenated. In the example stanza, the effective file name that is used for input to awsoma is `/media/ROM/xyz.tdf`. The slash (/) after ROM can be omitted, and a slash inserted before `xyz.tdf`, which results in the same effective file name.

Releases of ISV zPDT at the time of writing expanded the possible formats to include the following ones:

```
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 ISV 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 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 that is used to start th ISV zPDT operation. The last example depends on the keyword `TAPES` to indicate that data files are relative to the directory above `TAPES`.

The variable portion of the device statement can 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**.

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

[34] 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.

3.6.12 The awsctc device manager

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

```
[manager]
name awsckd 5432
device E40 3088 3088 ctc://192.168.1.81:3088/E42
                                 |        |   |
                                 |        |   + remote device number
                                 |        + remote port number
                                 + remote IP address
```

Multiple devices can be defined for this device manager. Chapter 16, “Channel-to-channel” on page 323 describes the setup and usage of this device manager.

4

ISV IBM Z Program Development Tool commands

Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) commands are entered as normal Linux command-line interface (CLI) commands in a Linux terminal. If ISV zPDT is running (that is, the IBM zSystems function is operating), then ISV zPDT commands that are directed to IBM zSystems must be entered in a Linux window that is owned by the Linux user ID that started the IBM zSystems function.[1] This situation is not normally an issue unless multiple ISV zPDT instances are running, each under a separate Linux user ID.

The commands that are described in this chapter are the "native" ISV zPDT commands. The Information Technology Company (ITC) offers a packaged ISV zPDT system as an ITC product known as *uPDT*. All the commands that are described in this chapter also apply to uPDT systems. In addition, ITC also provides a GUI to automate the common startup, shutdown, backup, restore, and other important processes. The GUI operation is not described in this publication, but more information is available from ITC.[2]

There is a considerable number of ISV zPDT commands that are available, and all of them are described in this chapter. As a practical matter, many commands are for specialized functions and seldom used in most ISV zPDT installations.

This chapter contains a few commands that are relevant only for IBM Z Development and Test Environment (ZD&T) systems, and they are included here for completeness. The IBM ZD&T product also offers more commands and utilities (not listed in this publication), which are described in the IBM ZD&T documentation.

[1] The same Linux user who issued the **`awsstart`** command must enter any additional ISV zPDT commands that affect that instance of IBM zSystems operation.

[2] Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 can be contacted at sales@itconline.com.

4.1 Command summary

Table 4-1 lists the commands, whether zPDT must be operational to use the command, how common the command might be, whether the command is primarily for IBM ZD&T systems, and whether Linux root authority is required to issue the command. The remainder of this chapter describes each command in detail. The purpose of this rather lengthy table is to provide a convenient way to check common questions, such as “Which commands do I need?” or “Which commands are only for IBM ZD&T?” or “Which commands really require Linux root authority?”

Table 4-1 zPDT command summary

Command	**zPDT must be operational**	**Usage**	**Mostly for IBM ZD&T**	**Linux root authority needed**
acptool	Required	Very uncommon		
alcckd	Optional	Common		
alcfba	Optional	Uncommon		
ap_create	Required	Obscure		
ap_destroy	Required	Obscure		
ap_query	Required	Obscure		
ap_von	Required	Obscure		
ap_voff	Required	Obscure		
ap_vpd	Required	Obscure		
ap_zeroize	Required	Obscure		
aws_bashrc	Unlikely	Common for ISV zPDT installation steps		
aws_sysctl	Unlikely	Common for ISV zPDT installation steps		Yes
aws_findlinuxtape	Optional	Uncommon		
aws_tapeInit	Optional	Uncommon		
aws_tapeInsp	Optional	Uncommon		
awsckmap	Usually not	Common		
awsin	Required	Uncommon		
awsmount	Required	Fairly common		
awsstart	No	Common		
awsstat	Required	Fairly uncommon		
awsstop	Required	Common		
card2tape	Optional	Uncommon		
card2txt	Optional	Uncommon		
checkLinux.sh	Optional	Uncommon		
ckdPrint	Optional	Fairly uncommon		
clientconfig	Optional	Partly common		Possible
clientconfig_authority	Optional	Uncommon		Yes
clientconfig_cli	Optional	Uncommon		Yes
cpu	Required	Fairly uncommon		
d	Required	Fairly uncommon		

Command	zPDT must be operational	Usage	Mostly for IBM ZD&T	Linux root authority needed
`dflt (might not be present)`	Required	Uncommon		
`display_gen2_acclog`	Optional	Uncommon	Yes	
`fbaPrint`	Optional	Fairly uncommon		
`find_io`	Optional; best	Common		
`gen2_init`	No	Fairly uncommon	Yes	Yes
`hckd2ckd`	Optional	Fairly uncommon		*a
`hfba2fba`	Optional	Uncommon		
`interrupt`	Required	Fairly uncommon		
`ipl`	Required	Common		
`ipl_dvd`	Required	Uncommon		
`ldk_server_config`	No	Uncommon	Yes	Yes
`listVtoc`	Best not	Uncommon		
`loadparm`	No	Very uncommon		
`managelogs`	No	Very uncommon		
`memld`	Required	Uncommon		
`mount_dvd`	Required	Uncommon		
`msgInfo`	Optional	Fairly common		
`oprmsg`	Required	Common		
`pdsUtil`	Best not	Uncommon		
`query`	Required	Fairly common		
`query_license`	Optional	Uncommon	Yes	
`rassummary`	Optional	Very uncommon		
`ready`	Required	Fairly common		
`request_license`	Optional	Uncommon	Yes	Yes
`restart`	Required	Fairly common		
`scsi2tape`	Optional	Uncommon		
`SecureUpdateUtility`	No	Not used. Integrated into z1090_token_update.	See text	Maybe yes
`SecureUpdate_authority`	Optional	Uncommon		Yes
`senderrdata`	Deleted	This command deleted		
`serverconfig`	Best not	Uncommon	Yes	
`serverconfig_cli`	Best not	Uncommon	Yes	Yes
`settod`	No	Fairly common		
`snapdump`	Deleted	This command deleted		
`st`	Required	Fairly common		
`start`	Required	Fairly uncommon		
`stop`	Required	Fairly uncommon		
`startstatus`	Required	Fairly uncommon		

Command	zPDT must be operational	Usage	Mostly for IBM ZD&T	Linux root authority needed
stpserverstarrt **stpserverstop** **stpserverquery**	-----	See Chapter 18, "Server Time Protocol" on page 347. Very uncommon		
sys_reset	Required	Uncommon		
tape2file	Optional	Uncommon		
tape2scsi	Optional	Uncommon		
tape2tape	Optional	Uncommon		
tapeCheck	Optional	Uncommon		
tapePrint	Optional	Uncommon		
token	Required	Common		
txt2card	Optional	Uncommon		
uimcheck	Optional	Fairly common		
uimreset	No	Fairly common		Yes
uimserverstart	Usually no	Uncommon		
uimserverstop	Usually no	Uncommon		
update_license	Usually no	Fairly uncommon	Yes	Yes
Z1090_ADCD_install	Usually no	Common		
Z1091_ADCD_install	Usually no	Common	Yes	
z1090_token_update	Usually no	Common		Yes
z1091_token_update	Usually no	Common	Yes	Yes
Z1090_removeall	No	Uncommon		Yes
z1090instcheck	Optional	Common		
z1090term	Optional	Uncommon		
ztrace	Yes	Fairly uncommon		
z1090ver	Usually no	Fairly common		
z1091ver	Usually no	Fairly common	Yes	
zPDTSecureUpdate	Usually no	Fairly common	Both 1090 and 1091	Yes

a. Linux root authority not needed, but special authority is needed to install an associated server program on the remote z/OS system.

4.2 ISV zPDT Linux command details

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

Most of the commands have a **help** option, which is invoked usually by an **-h** operand.[3] This operand is not shown in the following descriptions because it is the same for almost all commands and adds needless bulk to the command descriptions. The same help information can be obtained by running a Linux **man** command by using the ISV zPDT command name as the operand, as in the following example:

man awsstart	Requests the man pages for the **awsstart** command.
awsstart -h	Displays the same man pages.

The dollar sign ($) or number sign (#) in examples (and in many examples throughout this publication) represent the Linux prompts.

Some ISV zPDT commands require Linux root authority. The ISV zPDT program files (containing the ISV zPDT commands) are not normally in the Linux PATH for root. Depending on how you obtain root authority, this situation might be a problem. You can change to the `/usr/z1090/bin` directory (where the ISV zPDT program files are) and prefix the command with ./ ("dot slash"), or you might enter the complete path name to the command, such as **/usr/z1090/bin/uimreset**. The examples in this publication that illustrate root operation assume the use of the **su** command without the "dash" operand, which retains the PATH of the original user ID. However, this technique might not be appropriate for all users.

4.2.1 The acptool command

This command has restricted usage, and it is intended for users with extensive knowledge of the Common Cryptographic Architecture (CCA) operation of the crypto co-processor access points. This command has the following parameters:

[--dir-xxxx]	Indicates a directory other than CSUSRDI to read Security-Relevant Data Items (SRDI).
[--acps=0xAAA]	Comma-separated list of Access Control Points (ACPs) for certain commands.
[--role="roleid"]	Role identifier for certain subcommands.
[--list-roles]	List roles in `C_ROLES.srd`.
[--show-role]	Show role properties.
[--del-role]	Delete role (you must also specify **--role**).
[--load-role]	Create role (you must also specify **--role**).
[--disable-acps]	Set specified ACPs to 0 (specify **--role** and **--acps**).
[--enable-acps]	Set ACPs to 1 (specify **--role** and **--acps**).
[--query-acps]	Query ACPs (specify **--role** and **--acps**).

On normal IBM zSystems servers, a Trusted Key Entry (TKE) interface is used for functions like the ones that are listed here.[4] TKE usage is not available with zPDT, and this **acptool** command can provide a limited subset of some TKE functions. Again, do not use this command unless you have substantial experience in this area.

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

[4] A TKE provides many more functions than the ones that are listed here.

4.2.2 The adstop command

This command existed in early releases of zPDT. The command was removed, and replaced (in ISV zPDT GA11) with the **ztrace** command.

4.2.3 The alcckd command

The **alcckd** command creates (and formats) a Linux file that can be used as an emulated IBM 3380 or 3390 direct access storage device unit. The file is formatted to correspond to 3380 or 3390 tracks and cylinders in th CKD format, but is otherwise not initialized. A utility program (such as ICKDSF) must later be used to create a volume label, Volume Table of Contents (VTOC), and so on. A standard model (3380-1, 3380-2, 3380-3 or 3390-1, 3390-2, 3390-3, or 3390-9) can be specified to establish the size of the emulated device, a "non-standard" model number might be used,[5] or a specific number of cylinders can be specified. ISV zPDT does not need to be operational when using this command. (ISV zPDT might need to be restarted, with an updated device map (devmap) to use the newly created CKD device.)

Here is the command syntax:

```
alcckd file-name { -ddevice-type [-snumber-of-cylinders] [-q] }
                 { -r                                          }
                 { -rs                                         }
                 { -rf                                         }
                 { -ve | -vr | -vc | -vi | -vd }
                 { -f4 }
```

- **file-name** is a Linux file name.
- **-ddevice-type** is a device type, optionally with a model number. The following list presents the standard IBM 3380 and 3390 sizes. If no model number is specified, you must specify the number of cylinders with the **-s** parameter.
  - **-d3380-1** is device type 3380 with 885 cylinders.
  - **-d3380-2** is device type 3380 with 1770 cylinders.
  - **-d3380-3** is device type 3380 with 2655 cylinders
  - **-d3390-1** is device type 3390 with 1113 cylinders.
  - **-d3390-2** is device type 3390 with 2226 cylinders.
  - **-d3390-3** is device type 3390 with 3339 cylinders.
  - **-d3390-9** is device type 3390 with 10017 cylinders.

  The **type** field can specify a non-standard model number for 3390 devices. This model number is multiplied by 1113 to determine the number of cylinders to allocate. For example, **-d3390-20** allocates a 3390 with 20*1113 = 22260 cylinders.
- **-snumber-of-cylinders** is an alternative to specifying a non-standard model number. It determines the number of cylinders to be created. The maximum size is 65520 cylinders for a "normal" 3390, and an extended address volume (EAV) 3390 can be much larger.
- **-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.

[5] The non-standard sizes are intended only for 3390 devices.

- **`-rf`** performs the **`-rs`** function and reinitializes any emulated tracks with incorrect formats. The data content of that track is lost.
- **`-q`** invokes quiet mode with no output messages to the Linux terminal.
- **`-ve`**, **`-vr`**, **`-vc`**, **`-vi`**, and **`-vd`** are related to versioning. These parameters are described in Chapter 13, "Additional ISV IBM Z Program Development Tool notes" on page 273.
- **`-f4`** formats the new volume in 4 K blocks. Some Linux distributions (for IBM zSystems) cannot format a CKD volume, so this option is for use with those distributions.

Early releases of ISV 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.

If more than 65520 cylinders (for a 3390) are specified, an EAV is produced. The number of cylinders in an EAV must be an integral multiple of 1113.

The return values are as follows:

`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 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.

Here are some examples of the command's usage:

`$ alcckd /z/WORK01 -d3390-3`	Create an emulated 3390-3 volume.
`$ alcckd /z/MYVOL1 -d3390-20`	20*1113 cylinders.
`$ alcckd /tmp/222222 -d3390 -s100`	Create a small 3390 volume of 100 cylinders.
`$ alcckd /z/WORK01 -rs`	Verify the format of the CKD volume.

The **`-r`** set of flags displays information about a volume. One line of the information indicates whether initial program load (IPL) text is present. When an emulated volume is initialized by ICKDSF, a minimal IPL text is automatically loaded. This default IPL text loads a disabled wait state with the Program Status Word (PSW) code x'0F', which means no "real" IPL text is present. However, the presence of this small default IPL text causes **`alcckd`** to indicate that IPL text exists. The only time **`alcckd`** indicates that IPL text does not exist is if the emulated volume was not initialized by ICKDSF.

Tip: Do not create emulated 3380 devices unless you have a specific need for them.

4.2.4 The alcfba command

The **alcfba** command creates (and formats) a Linux file that can be used as an emulated IBM 9336 direct access storage device unit. The file is formatted to correspond to the fixed blocks of a 9336 device, and a volume name can be assigned. A standard model (9336-1 or 9336-2) can be specified to establish the size of the emulated device, or a specific number of blocks can be specified to create a nonstandard size. (Fixed-block devices compatible with 9336 drives also can use these emulated volumes.) ISV zPDT does not need to be operational when using this command. (ISV zPDT can be restarted with an updated devmap to use the newly created Fixed Block Architecture (FBA) device, or you could use the **awsmount** command to mount the new volume on an existing FBA device in a running ISV zPDT system.)

Here is the command syntax:

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

- **file-name** is the Linux file name for the emulated volume.
- **-ddevice-type**:
  - **-d9336** is a device type, and its size is set by the **-s** parameter.
  - **-d9336-1** is a device type, and its size is 920,115 blocks.
  - **-d9336-2** is a device type, and its size is 1,672,881 blocks.
- **-ssize** is the size (in decimal) of the emulated volume:
  - **-snnnB** specifies the number of 512 K 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 6 characters and automatically converted to uppercase.
- **-c** changes the volser of an existing FBA volume.
- **-q** sets quiet mode with no output messages that are sent to the Linux terminal.
- **-r** display the attributes of an existing FBA volume.

Earlier releases of ISV 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.

The Return values are as follows:

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 exists.
16	File not found or the file name is not valid.

`17`	Drive is not ready.
`19` or `20`	Disk is not valid.

Command examples are as follows:

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

4.2.5 The ap_create command

The **`ap_create`** command dynamically creates an emulated cryptographic processor. ISV zPDT must be started when this command is used.

Here is the command syntax:

```
ap_create -a n
```

`n` is the number of the coprocessor. The number is 0 - 15.

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

4.2.6 The ap_destroy command

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

Here is the command syntax:

```
ap_destroy -a n
```

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

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

4.2.7 The ap_query command

The **`ap_query`** command displays the status of emulated cryptographic coprocessors. ISV zPDT must be started when this command is used.

Here is the command syntax:

```
ap_query
ap_query -a n
```

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

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

4.2.8 The ap_von and ap_voff commands

The **ap_von** and **ap_voff** commands vary emulated cryptographic co-processors (or domains) online or offline. ISV zPDT must be started when this command is used.

Here is the command syntax:

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

- **n** is the number of a cryptographic coprocessor.
- **y** is the number of a domain within the specified coprocessor.

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

4.2.9 The ap_vpd command

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

Here is the command syntax:

```
ap_vpd -a n
```

n is the number of a defined cryptographic coprocessor.

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

4.2.10 The ap_zeroize command

The **ap_zeroize** command erases (zeros) the content of a specified emulated cryptographic co-processor or a subset of a co-processor. ISV zPDT must be running when this command is used.

Here is the command syntax:

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

- **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 co-processor is used (or when one is zeroed), it must be reinitialized. This process is normally done with the Integrated Cryptographic Service Facility (ICSF) utility, as described in Chapter 17, “Cryptographic usage” on page 331.

4.2.11 The attn command

The **attn** command creates a simulated attention interrupt from a device. ISV zPDT must be operational when using this command.

Here is the command syntax:

```
attn device-number
```

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

The meaning of an attention interrupt varies depending on the device type. In typical ISV zPDT operation, this command is probably not used.

A command example is as follows:

```
$ attn 590
```

4.2.12 The aws_bashrc and aws_sysctl commands

These commands can be used during ISV zPDT installation to bypass making tedious manual changes to Linux files. ISV zPDT normally is not operational when using these commands. These two shell scripts are in `/usr/z1090/bin`, along with all the other ISV zPDT command files. However, at the time that 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 suggested modifications to `/etc/sysctl.conf` and then runs **/sbin/sysctl**. The **aws_sysctl** command must run with root authority. The statements that this command adds to `/etc/sysctl.conf` are appropriate for many ISV zPDT users, but it might need to be manually modified for larger zPDT instances. For more information, see Chapter 5, “ISV IBM Z Program Development Tool installation” on page 125.

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

4.2.13 The aws_findlinuxtape command

This command lists any SCSI tape drives that are connected to the underlying personal computer (PC).

Here is the command syntax:

```
aws_findlinuxtape
```

This command lists both the `st` and `sg` numbers that are currently associated with each tape drive that is found. For more information about the device identifiers that are used with SCSI tape drives, see “The awsscsi device manager” on page 305. ISV zPDT does not need to be operational when using this command. The sg number can change when Linux starts.

4.2.14 The aws_tapeInit command

This command creates an emulated tape (a Linux file in awstape format) with a standard label.

Here is the command syntax:

```
aws_tapeInit volser file-name
```

The **volser** is the volume serial number of the new tape, and **file-name** is the Linux file name for the emulated tape volume. The **volser** must be 6 characters, letters, or numbers with no special characters. (This requirement is slightly more restrictive than volume serial numbers that are created by other means.) The **volser** is automatically converted to uppercase Extended Binary Coded Decimal Interchange Code (EBCDIC). ISV zPDT does not need to be operational when using this command.

Here are some examples of this command:

$ aws_tapeInit 222222 /z/tapexxyyzz	Creates volser 222222.
$ aws_tapeInit tape01 /tmp/mywork/TAPE01	Creates volser TAPE01.

4.2.15 The aws_tapeInsp command

This command examines a Linux file. If the file appears to be an emulated tape volume in awstape format, it is examined for standard header labels. Basic information from the standard header labels is displayed. ISV zPDT does not need to b operational when using this command.

Here is the command syntax:

```
aws_tapeInsp file-name
```

The **file-name** is the Linux file name of an emulated tape volume. Simple tests are made to determine whether the file is in awstape format. If it appears to be in awstape format, it is checked for VOL1, HDR1, and HDR2 labels. The data set name and dataset control block (DCB) parameters are displayed if they are present. The emulated tape volume is not scanned other than the first three records, and subsequent labels on the tape are not inspected.

Here is an example of the command:

```
$ aws_tapeInsp /z/tape01.aws
   volser: TAPE01
   DSN: SMF30.EXTRACT
   RECFM: FB  LRECL: 100  BLKSIZE: 10000
```

4.2.16 The awsckmap command

The **awsckmap** command validates the content and format of a devmap by reporting any errors that are found. ISV zPDT does not need to be operational when using this command.

Here is the command syntax:

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

- **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 or devices in the devmap.
- **--mgr** lists the device managers that are required by this devmap.
- **--dev** lists detailed device information from the devmap.

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

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

4.2.17 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. (3215 devices are rare today, and this command is seldom used.) ISV zPDT must be operational when using this command.

Here is the command syntax:

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

- **dev-address** is the address (device number) from the devmap.
- **'text'** is the message that is sent to the 3215.
- **-a** indicates that an attention interrupt should be sent, but no text.

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

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 as follows:

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

4.2.18 The awsmount command

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

Here is the command syntax:

```
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 }
```

- **dev-address** is the device address from the devmap.
- **-b** or **--bsf** backspaces over one tape mark on an emulated 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, which removes any previous Linux file association with the device.
- **-r**, **--ro**, or **--readonly** makes the emulated device read-only.
- **-w**, **--rw**, or **--readwrite** makes the emulated device read/write.
- **-o** or **--replace** replaces the existing file association with a new file association (like 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**, **--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 can be used with SCSI-attached tape drives. The correct **awsmount** functions can 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 usage of awsmount (by using 580 as a typical device number) are as follows:

- For tape drives (emulated or SCSI):

`$ awsmount 580 -q`	Query the currently mounted file.
`$ awsmount 580 -m  /tmp/tapevol/123456`	Mount the emulated volume.
`$ awsmount 580 -o  /z/654321`	Replace the mounted volume
`$ awsmount 580 -u`	Unmount the current volume.
`$ awsmount 580 -x` or **`--run`**)	Unmount the current volume.
`$ awsmount 580 -b`	Back space over the tape mark.
`$ awsmount 580 -f`	Forward space over the tape mark.
`$ awsmount 580 -s`	Rewind the tape volume.
`$ awsmount 580 -t`	Write the 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 the currently mounted file.
`$ awsmount 180 -m /tmp/oma/11111`	Mount the emulated volume.
`$ awsmount 180 -o /z/oma/dosvol`	Replace the mounted volume.
`$ awsmount 180 -u`	Unmount the current volume.
`$ awsmount 180 -x` or **`--run`**	Unmount the current volume.
`$ awsmount 180 -b`	Back space over the tape mark.
`$ awsmount 180 -f`	Forward space over the tape mark.
`$ awsmount 180 -s`	Rewind the tape volume.

- Disks and printers (using 300 and 00E device numbers):

`$ awsmount 300 -q`	Query the mounted file name.
`$ awsmount 300 -m  /z/LOCAL1`	Mount the emulated volume.
`$ awsmount 00E -m  /tmp/print1`	Printer output file.
`$ awsmount 300 -o  /z/LOCAL2`	Replace the mounted volume.
`$ awsmount 300 -u`	Unmount the current volume.

- aws3270 (local 3270 sessions, for example, device number 702):

`$ awsmount 702 -q`	Query the tn3270 client.
`$ awsmount 702 -d`	Force a disconnect.

- awsscsi (connect SCSI tape drives, for example, by using device number 580):

`$ awsmount 580 -m /dev/sg3`	Connect the SCSI tape drive.

4.2.19 The awsstart command

The **awsstart** command starts the ISV zPDT operation by creating an IBM zSystems environment.

Here is the command syntax:

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

- **--noosa** creates an ISV zPDT environment without any Open Systems Adapter (OSA) components.
- **--map** is optional before the devmap file name.
- **file-name** is the name of a devmap file.
- **--clean** deletes all previous logs and traces.
- **--localtoken** causes a local USB token to be used, and ignores any remote license servers.

Using the **--noosa** parameter is unusual, and it should be used *only* at IBM direction. ISV zPDT maintains various logs and traces in the ~/z1090/logs directory. This directory is a subdirectory of the user ID that starts ISV zPDT. The contents of the logs directory can grow over time. If no ISV zPDT problems are under investigation, using the **--clean** parameter ensures that only currently relevant logs and traces (from the ISV zPDT instance just being started) appear in the directory.[6]

An ISV zPDT system can 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 can be used. For more information, see Chapter 8, "ISV IBM Z Program Development Tool licenses" on page 179.

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

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

4.2.20 The awsstat command

The **awsstat** command queries the status of emulated I/O devices. ISV zPDT must be operational when using this command.

Here is the command syntax:

```
awsstat [-i [n-seconds]] [device-list] [-s [pid|addr|subchan|mgr|busy]]
                                       [--sort [pid|addr|subchan|msg|busy]]
```

- **-i n-seconds** indicates that 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 **device-list** is not provided, all defined emulated devices are listed. A range of device numbers can be specified, or the name of a device manager.
- **-s** or **--sort** can be used to sort the **awsstat** output according to the indicated criteria. By default, it is listed in subchannel order.

device-list can use 3- or 4-digit hexadecimal operands, which are the device numbers ("addresses") that are defined in the current devmap. The output display for emulated disk devices includes the current head position (cylinder and track) on the device.

[6] ISV zPDT automatically manages the logs, so using the **--clean** option is not normally necessary.

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 terminates the command.

The defined return values are as follows:

`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 the device status.

An example of the command and the resulting output is as follows:

```
$ awsstat
Config file: /home/ibmsys1/aprof9 3270port: 3270  Instance: ibmsys1
DvNbr S/Ch --Mgr--- IO Count --PID--  ------Device information------------
0700     0 AWS3274     141     4315   IP-::1
0a80     5 AWSCKD    335300    4449   /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 IO Count field is the number of SSCH[7] commands that are issued to the device since the last IPL. The process ID (PID) number is the base Linux process number that is emulating the device.

`$ awsstat a80-a85`	A range of device numbers
`$ awsstat awsckd`	All devices that are owned by this device manager

4.2.21 The awsstop command

The **`awsstop`** command ends the ISV zPDT operation. This operation ends abruptly with no warning to the IBM zSystems operating system. You normally shut down your IBM zSystems operating system in an orderly manner before using **`awsstop`**.

Here is the command syntax:

```
awsstop
```

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

```
$ awsstop
```

4.2.22 The card2tape command

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

Here is the command syntax:

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

- **`-c`** or **`--compress`** causes the output awstape file to be compressed.
- **`-a`** or **`--ascii`** indicates that the input file is ASCII and causes the output to be converted to EBCDIC.

[7] SSCH is the Start Subchannel instruction for IBM zSystems. Counting I/O operations can be a fuzzy topic due to command chaining in channel programs or queued I/O devices.

The compression option saves space in the emulated output file, but it is not compatible with other platforms that can use awstape files. The output is in unblocked 80-byte records, with blanks appended to input records if necessary. Standard tape labels are not created. If the input file length is not a multiple of 80, then the **-a** option must be used.

The default conditions for ASCII to EBCDIC conversion are the same as used for the awsrdr device manager. The **-a** or **--ascii** parameters can be used to force conversion. The EBCDIC to ASCII translation table that is used cannot be changed.

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

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

4.2.23 The card2txt command

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

Here is the command syntax:

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

- **input-file** is an EBCDIC file that must be an exact multiple of 80 bytes.
- **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 New Line (NL) character is added, as in a Linux text file. The EBCDIC to ASCII translation table used cannot be changed.

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

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

4.2.24 The checkLinux.sh command

The **checkLinux.sh** command displays (in the Linux window) the name of the current Linux distribution. This command is implemented as a simple shell script and might be expanded in future zPDT releases. It is primarily used internally by other zPDT administrative commands, but can be useful in various debugging or user assistance situations. zPDT does not need to be running when this command is used.

4.2.25 The ckdPrint command

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

Here is the command syntax:

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

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, which are entered as four decimal numbers that are separated by blanks. The numbers, in sequence, are as follows:

- 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. To terminate the program, press Ctrl+C. The count, key, and data fields are shown for each block on the track (or tracks) 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 shown here:

```
$ 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
```

4.2.26 The clientconfig command

The **clientconfig** command provides an interactive menu function to help configure a remote license server and a Unique Identity Manager (UIM) that provides consistent IBM zSystems serial numbers. This command must be run as root[8] unless the **clientconfig_authority** function is used to bypass this limitation. ISV zPDT does not need to be operational (and probably would not) when using this command.

Here is the command syntax:

```
clientconfig
```

A description of the command's usage is in Chapter 8, "ISV IBM Z Program Development Tool licenses" on page 179.

4.2.27 The clientconfig_authority command

The **clientconfig_authority** command adds a Linux user ID or removes a Linux user ID from a list of user IDs that can issue the **clientconfig** command. Normal usage of **clientconfig** requires the user to operate as root. The **clientconfig_authority** command allows the installation to avoid using root when changing license server configurations. The **clientconfig_authority** command itself must be run as root,[9] but it is used only once for a given user ID. ISV zPDT does not need to be operational when using this command.

After a user ID is authorized with this command, they can use the Linux **sudo** command to run **clientconfig**.

[8] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from /usr/z1090/bin.

[9] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from /usr/z1090/bin.

Here is the command syntax:

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

- **-a** adds the indicated user ID to the list of user IDs that are allowed to issue the **clientconfig** command.
- **-d** removes the indicated user ID from this list.

A command example is as follows:

```
# clientconfig_authority -a ibmsys1
```

4.2.28 The clientconfig_cli command

This command provides a Linux CLI (non-interactive) for **clientconfig**. The CLI may be used in a Linux script. You must be root to use this command.[9] ISV zPDT does not need to be operational when using this command.

Here is the command syntax:

```
clientconfig_cli [-g1s1 xxx] [-g1s2 xxx] [-g2s1 xxx]  [-g2s2 xxx]
                                [-usc xxxx] [-usm y|n] [-l]
```

- **-g1s1** specifies a Gen1 license server and uses a numeric IP address or a domain name.
- **-g1s2** specifies a backup Gen1 license server and uses a numeric IP address or a domain name.
- **-g2s1** specifies a Gen2 license server and uses a numeric IP address or a domain name.
- **-g2s2** specifies a backup Gen1 license server and uses a numeric IP address or a domain name.
- **-usc** specifies a UIM server (numeric or domain name). It defaults to the same address as the license server (the Gen2 server, if both Gen2 and Gen1 are specified).
- **-l** requests a display of the currently configured license servers.

There must be a space between a “flag” and the operand of that flag.

A Gen1 or Gen2 server (or both) *must* be specified.

Command examples are as follows:

- **# clientconfig_cli -g1s1 my.remote.licenses.net**
- **# clientconfig_cli -g2s1 192.168.1.107 -g2s2 123.321.111**
- **# clientconfig_cli -l**

4.2.29 The cpu command

The **cpu** command selects the default CP that is the target for subsequent ISV zPDT commands. IBM zSystems Integrated Information Processor (zIIP), IBM zSeries Application Assist Processor (zAAP), and IBM Integrated Facility for Linux (IFL) processors are considered CPs for this function. ISV zPDT must be operational to use this command.

Here is the command syntax:

```
cpu cp-address
```

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, zAAP, or IFL) that is defined in the **processors** statement of the devmap. The default target is CP number zero. Each CP has its own registers, active address space, and so on. This command typically is used to examine registers and memory in a particular CP.

The defined return codes are as follows:

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 as follows:

```
$ cpu 1          (Select second CP as the default CP, which is CP number 1.)
$ stop           (Place the default CP into the stopped state.)
$ d psw          (Display the PSW of the default CP.)
$ start          (Start the default CP again.)
```

4.2.30 The d command

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

Here is the command syntax:

```
d {r                                                        }
  {p | psw                                                  }
  {pfx                                                      }
  {g | gn                                                   }
  {y | yn                                                   }
  {yc                                                       }
  {x | xn                                                   }
  {z | zn                                                   }
  {vphex-addr[.]hex-len | [ ]hex-len                        }
  {vshex-addr[.]hex-len | [ ]hex-len                        }
  {vhhex-addr[.]hex-len | [ ]hex-len                        }
  {vahex-addr[.]hex-len | [ ]hex-len  access-reg            }
  {[t]hex-addr[.]hex-len | [ ]hex-len]                      }
  {vr | vr[m]        (m=0-31)                               }
   {vc                                                     }
   {lso                                                      }
  {hn                                                       }
```

- **r** displays the current architecture mode.
- **p** or **psw** displays the current PSW.
- **pfx** displays the prefix register.
- **g** or **gn** displays the contents of the general-purpose registers. If a particular register is not specified (by the **n** parameter), then all of them are displayed.
- **y** or **yn** displays floating point registers.
- **yc** displays the floating point control register.
- **x** or **xn** displays control registers.
- **z** or **zn** displays access registers.

- **h** displays subchannel information for subchannel *n*. This option is primarily for IBM usage.
- **lso** displays the leap second information block.
- **hex-addr** is an address in memory.
- **.hex-len** is the amount of memory to be displayed (in hexadecimal). If a period precedes the length, the period must immediately follow the address. A blank separating the address and length (instead of a period) can be used. The length is in hexadecimal.
- **vp** displays the primary virtual memory. The **vp** prefix can be shortened to **v**.
- **vs** displays the secondary virtual memory.
- **vh** displays the home address space virtual memory.
- **va** displays the virtual memory through an access register, which must be specified.
- **vr** displays one or more vector registers in a single, long hex field.
- **vc** displays the floating point control register.
- **access-reg** is the number of an access register.
- **t** (just before a real address) indicates that both hex and character displays are wanted.

A memory address that is not prefixed with **vp**, **vs**, **vh**, or **va** displays data at the IBM zSystems real memory address.[10] 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 that is 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.

A virtual address is meaningful only if an address space is active at the instant of the display.[11] 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 run, with the appropriate virtual memory translation tables active.

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

The return values are as follows:

0	Command complete.
30	No arguments are specified.

Examples of usage are as follows:

$ d psw	Displays PSW.
$ d g2	Displays the contents of general-purpose register 2.
$ d 461244 32	Displays x’32’ bytes at real address x’461244’.
$ d 461244.C0	Displays x’c0’ bytes at the indicated address.
$ d v458332 100	Displays x’100’ bytes at the indicated virtual address.
$ d vr	Displays all 32 vector registers.
$ d t150 10	Displays 16 bytes at read address 150, with EBCDIC.

[10] Normally, the real and absolute addresses are the same. If the real address is reassigned by the prefix register, then both the real and absolute addresses are displayed.

[11] If the appropriate segment tables (or region tables) for the target address space are indicated by the appropriate control registers for the default CP.

4.2.31 The dflt command

This command enables (or disables) the Deflate Facility indicator, and this facility refers to the DFLTCC instruction.

Here is the command syntax:

```
dflt [-Y | -y | -N | -n | <no operand> ]
```

- The **-Y** or **-y** operand enables the Deflate Facility indicator.
- The **-N** or **-n** operand disables the Deflate Facility indicator.

If no operand is present, the current facility setting is reversed. If needed, the **dflt** command must be used before any IBM zSystems operating system undergoes an IPL. The command displays `0` or `1`, which indicates the current indicator setting. Using the command while the IBM zSystems operating system is running might not be effective.

The DFLTCC instruction is included in zPDT GA11 releases (in synchronous mode only), and the Deflate Facility indicator is disabled by default. This odd combination is due to a rare problem with the DFLTCC instruction that had not been corrected when GA11 was released. The **dflt** command changes the facility indicator, so disabling this indicator does not disable the DFLTCC instruction itself. A best practice is to test the indicator first before directly using the instruction.[12] In general, this instruction is used by operating system routines and not by direct application programming.

The **dflt** command might be removed in future zPDT versions.

4.2.32 The display_gen2_acclog command

The **display_gen2_acclog** command displays the access log that is maintained internally by a Gen2 license server. The displayed data is sent to Linux stdout and can be redirected to a normal Linux file. Displaying the log data does not delete it. The Gen2 license server “trims” the log at intervals. This command is functional only on a Gen2 license server, and it cannot be used on a Gen2 client system.

Here is the command syntax:

```
display_gen2_acclog
```

There are no operands. At the time of writing, Gen2 tokens or software licenses are not available for ISV zPDT systems.

4.2.33 The fbaPrint command

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

Here is the command syntax:

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

emulation-file-name is the name of the Linux file that contains the FBA volume.

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

[12] The STFLE instruction can be used to examine the state of “facility” indicators.

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

```
$ fbaPrint /z/XYZ123
0  1                        (Dump two blocks.)
```

4.2.34 The find_io command

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

Here is the command syntax:

```
$ find_io
FIND_IO for "ibmsys1@linux-8jfl"
        Interface        Current          MAC               IPv4             IPv6
 Path   Name             State            Address           Address          Address
------  ---------------- ---------------- ----------------- ---------------- --------
  F0    eth0             UP, NOT-RUNNING  50:7b:9d:ac:73:45  *               *
  F8    wlan0            UP, RUNNING      e4:b3:18:c9:11:a2  192.168.1.108   xxxxxx
  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  *               *
  ...

        Interface                        Current Settings
 Path   Name             RxChkSum         TSO     GSO   GRO  LRO    RX VLAN      MTU[13]
------  ---------------- ---------------- ----------------- ---------------- --------
  F0    eth0             On[14]           On[14]  On[14] On[14] Off   On[14]      1500
  F8    wlan0            Off              Off     On[14] On[14] Off   Off         1500
End of FIND_IO
```

A path (or CHPID) name is shown for most (but not necessarily all) local area network (LAN) interfaces. The paths have names such as F0, F1, and A0. Interface names are shown, for example, eth0, tap0, wlan0, and so on. A path name is normally specified for the awsosa device manager, but in some cases the interface name might be needed if no path name is shown by the **find_io** command. All Linux LAN interfaces, enabled or disabled, are detected, which might cause default path assignments to differ from previous ISV zPDT releases.

The other data that is shown (State, MAC address, IPV4, and IPv6 addresses) is informational only. The IP addresses apply only to the base Linux. z/OS (or another IBM zSystems operating system) also can address the interfaces with different IP addresses. All LAN interfaces that are known to Linux are shown, but some of them might not be relevant or tested for ISV zPDT usage. The MAC addresses for *tap* devices are artificial.

The **ethtool** parameters that are shown (**TSO**, **GSO**, **GRO**, **LRO**, and **RX VLAN**) can affect performance. The maximum transmission unit (MTU)[15] size that is shown is the Linux (not z/OS) MTU, but it should be coordinated with the z/OS (or z/VM or other operating system) MTU. For more information about these parameters, see Chapter 7, “Local area networks” on page 149.

[13] Enabling these functions might lead to poor ISV zPDT performance. For more information, see your ISV zPDT documentation.
[14] To Enable Jumbo Frame Support, this MTU value and the MTU value for the host operating system must be set to > 1500.
[15] MTU is the number of bytes that are sent as a single unit over the LAN.

4.2.35 The gen2_init command

This command installs (or removes) the Gen2 license interface modules. The command must be run with the root authority.[16]

Here is the command syntax:

```
gen2_init [--remove]
```

Earlier versions of this command can trigger a web search for certain 32-bit modules. The license interface modules that are supplied with ISV zPDT GA9 are 64-bit modules, so no search for them is needed. At the time of writing, Gen2 tokens and software licenses are not currently available for ISV zPDT.

4.2.36 The hckd2ckd and hfba2fba commands

These commands deal with the migration of direct access storage device volumes from a remote system to an ISV zPDT system. In this context, "migration" means "copy." The are two client commands that are used with the migration utilities:

hckd2ckd	Used with both z/OS and z/VM to migrate a CKD direct access storage device volume.
hfba2fba	Used only with z/VM to migrate an FBA direct access storage device volume.

ISV zPDT does need to be operational when using these commands. The general syntax of the client commands (entered on the Linux client machine by using a normal Linux CLI) is as follows:

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

- **host** is the TCP/IP name of the system with the matching server program. This name can be a dotted decimal address or a domain name that can be resolved by Linux TCP/IP.
- **:port** is a TCP/IP port number that is used by both the client and server programs. The number defaults to 3990. The port number does not always default correctly, so as a best practice, always include the port number in the command.
- **outfile** is a file name (on the current Linux system) where the migrated volume is placed (in awsckd or awsfba format).
- **-v** or **--volser** indicates the 3380 or 3390 volume (on the remote z/OS system) that is copied (migrated).
- **-u** or **--unit** indicates the address (device number) of the volume that is copied (migrated).

Either the **-u** or **-v** parameter must be supplied, but not both. These commands are described in Chapter 15, "Direct access storage device volume migration" on page 313.

Examples of commands that can be used to run the client are as follows:

- **`$ hckd2ckd 192.168.1.99:3990  /z/VOL123  -v VOL123`**
- **`$ hckd2ckd BIG.ZOS.ADDR:3990 /z/VOL678 -u A8F`**
- **`$ hckd2ckd 192.168.1.99:3990 /z/host.WORK23 -v WORK23`**

[16] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from `/usr/z1090/bin`.

4.2.37 The interrupt command

The **interrupt** command creates an external interruption for a CP. ISV zPDT must be operational when using this command.

Here is the command syntax:

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

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

The effect of an external interrupt depends on the IBM zSystems operating system that is used. The return values are as follows:

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

Examples of usage are as follows:

$ interrupt	Interrupts the default CP.
$ interrupt 1	Interrupts CP number 1.

4.2.38 The ipl command

The **ipl** command starts the process of loading an operating system (or a stand-alone utility program). ISV zPDT must be operational when using this command.

Here is the command syntax:

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

- **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 8 characters that provides more information about the operating system that is loaded.
- **clear** causes IBM zSystems memory to be zeroed before loading the operating system.
- **gprparm** provides a string of characters that are inserted into the general-purpose registers (as EBCDIC characters that use 32-bit registers), starting with register 0, placing 4 characters in each register. The keyword can be entered as **gpr_parm**. The **gprparm** function was carried forward from much earlier systems and has no known use today.

The **ipl** function is started on the default CP (which must not be a zIIP or zAAP), which can be set by the **cpu** command. Using **parm-value** depends on the operating system that is used and how that operating system is configured. As a general statement, it is not necessary to run **clear** before loading an operating system.

The device that is indicated by **device-number** must have IPL text that is installed, which is normally done 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.

Generally. the program that undergoes the IPL tries to interact with a console, which might be an emulated 3270 session or the Hardware Management Console (HMC) (also known as the “hardware console”). Messages that are sent to the HMC console appear in the Linux CLI that issued the **awsstart** command to start ISV zPDT.[17] Such messages (in the Linux CLI) with the prefix `OPRMSG` and text can be sent to the HMC console by running the **orpmsg** command. For example:

```
OPRMSG: 12.13.14 HCPASK9205A To change to a FORCE start, enter FORCE
$ oprmsg force
OPRMSG: HCPSED6013A A CP read is pending
$ oprmsg                           (no operand is equivalent to pressing ENTER)
```

The command return values are as follows:

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

Examples of command usage are as follows:

- **$ ipl 580**
- **$ ipl 0a80 parm 0a82cs clear**

4.2.39 The ipl_dvd command

The **ipl_dvd** command emulates an IPL from a DVD from the HMC on a larger IBM zSystems. The DVD must contain files in a specific format for this function to be used. At the time of writing, the only known uses are with an optional form of z/VM system distribution, some Linux for IBM zSystems distributions, possibly a few other operating systems, and the Customized Offering Driver (COD) z/OS system. ISV zPDT must be operational for this command to be used.

Here is the command syntax:

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

- **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). Then, this 3270 is used as an HMC 3270 session. (At the time of writing, this function was not working with z/VM 6.2 and later. Instead, the **int3270port** function in the `[system]` stanza of the devmap can be used.) The usage of the **-c** or **--console** is deprecated. This option might 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 default IBM zSystems processor (normally the one that is listed first in the **processor** statement in the devmap) must be a standard CP. It cannot be an IFL, which is a departure from the standard IBM zSystems architecture.

This command does not require DVD usage. The `.ins` file and other associated files can be in a normal disk-resident Linux file system.

[17] For more information about how to handle these messages, see 13.5, “zPDT log files” on page 282.

The return values are as follows:

`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 the `.ins` file.

An example of usage is as follows:

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

4.2.40 The ldk_server_config command

This command can be used if a remote Gen2 license server is used to provide licenses to your client. It is a CLI alternative to the interactive **`clientconfig`** command. You must be root to use this command.[18] ISV zPDT does need to be (and probably is not) active when this command is used.

Here is the command syntax:

```
ldk_server_config  [server.url.address]
                     [-d]
                     [-L]
```

- **`server.url.address`** is the address of your Gen2 server. The address can be a URL (domain name) or a numeric IP address. Multiple addresses, which are separated by spaces, can be listed if you have alternative Gen2 servers.
- **`-d`** indicates that the currently active Gen2 server addresses will be listed.
- **`-L`** indicates that no remote Gen2 server will be used.

Here are some examples of this command:

- **`# ldk_server_config perf.lab.ibm backup.lab.ibm`**
- **`# ldk_server_config 192.168.1.107`**
- **`# ldk_server_config -d`**

This command is deprecated. The same functions can be done with the **`clientconfig_cli`** command. At the time of writing, Gen2 tokens and software licenses are not available for ISV zPDT systems.

4.2.41 The listVtoc command

The **`listVtoc`** command provides a detailed listing of the VTOC of a z/OS CKD volume. It assumes that the emulated CKD volume was initialized with a label and a z/OS compatible VTOC. This command can be used while ISV zPDT is operational, but it normally is used when ISV zPDT is not operational.

Here is the command syntax:

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

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

[18] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from `/usr/z1090/bin`.

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 as follows:

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

4.2.42 The loadparm command

The **loadparm** command sets an 8-character IPL parameter value that can be read by a special IBM zSystems instruction. This parameter is also known as a *load* parameter. *IPL* and *load* are used as synonyms in this context.

Here is the command syntax:

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

- **value** is the character string to set (up to 8 characters).
- **-d** or **display** displays the current value.

This value that is 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 ISV zPDT operation, and it is not retained across multiple ISV zPDT startups. This command is probably never used by normal ISV zPDT installations.

The return values are as follows:

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

Examples of the command usage are as follows:

- **$ loadparm 0A8200P**
- **$ loadparm -d**

4.2.43 The managelogs command

The **managelogs** command helps maintain summary, trace, and log files in the zPDT logs directory. Generally, 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 an IBM Business Partner) while investigating a potential zPDT problem. zPDT should not be operational when this command is used.

Here is the syntax of the command:

```
managelogs {file-name      }
           {-s snap-id     }    snap and snapid not used with current zPDT
           {-t date        }
```

- **`file-name`** removes the summary record and associated file.
- **`snap-id`** removes all summary records and files that are associated with the specified snap ID.
- **`date`** removes all summary records and files older than the indicated date. The date format is yyyy/mm/dd.

The **`rassummary`** command may be used to determine existing snap ID numbers, if any. The general creation of "snap" information is not used by ISV zPDT releases at the time of writing.

There are no return values for this command.

4.2.44 The memld command

The **`memld`** command is used to write the contents of a Linux file into IBM zSystems memory, starting at a specified address. ISV zPDT must be operational when using this command.

Here is the command syntax:

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

- **`file-name`** is a fully qualified Linux file name.
- **`address`** is an IBM zSystems hexadecimal address. The default is address zero.

Some Linux for IBM zSystems distributions can be installed by loading various files into IBM zSystems memory and then running an IBM zSystems **`restart`** function.

The return values are as follows:

`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 as follows:

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

4.2.45 The mount_dvd command

The **`mount_dvd`** command identifies the Linux mount point for a DVD (or CD) that will be processed as thought it were mounted in the DVD drive of an IBM zSystems HMC. ISV zPDT must be operational when using this command.

Here is the command syntax:

```
mount_dvd complete-path
```

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

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

An example of the command is as follows:

```
$ mount_dvd /run/media/ibmsys1/zVM_RSU_name/
```

4.2.46 The msgInfo command

The **msgInfo** command provides more information about ISV zPDT messages. ISV zPDT does not need to be active when this command is used.

Here is the command syntax:

```
msgInfo message-number
```

message-number is the number of an ISV zPDT message.

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

```
$ 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 is needed, but
AWSINF013I     if errors are present, then the DEVMAP cannot be used to start the
               system.
```

All message numbers are in the form `AWS`*`cccnnns`*:

- `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, or Terminal).

The message code that is 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 that is named **Z1090_MSG** to control message formatting.[19] It can be set to `FULL` (the default), `CODE` (prints only the message number and no text), `TEXT` (prints the message text and no code), and `SHORT` (drops the AWS prefix on the message number).

4.2.47 The oprmsg command

The **oprmsg** command provides input to IBM zSystems through the SCLP operator message interface. (This interface is also known as the *HMC* or the *hardware console*.) ISV zPDT must be operational when using this command.

Here is the command syntax:

```
oprmsg {text}
```

text is the message that is sent to the IBM zSystems operating system. If it contains any special characters (such as parentheses), the message should be enclosed in single quotation marks.

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

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

Using the **intASCIIport** or **int3270port** statements in the devmap can interfere with "HMC console" output to a Linux window and with the **oprmsg** command. These devmap statements should be used only if there is a specific requirement for them.

The **oprmsg** command is entered into a Linux CLI and processed by the Linux shell running that window, which means that the operands of the **oprmsg** command are examined by the Linux shell and might interpret or change or omit characters before passing the operand to the **oprmsg** program. For this reason, enclose the operand in single quotation marks, as shown in the following examples.

The return values are as follows:

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 the command usage are as follows:

- **$ oprmsg 'V CN(*),ACTIVATE'**
- **$ oprmsg 'V 700,CONSOLE'**
- **$ oprmsg 'D A,L'**
- **$ oprmsg m '* "hello Colin"'** (Results in `m * 'hello Colin '`.)

4.2.48 The pdsUtil command

The **pdsUtil** command is a Linux command that reads (or rewrites) members of a z/OS partitioned data set (PDS). z/OS is normally not operational when this command is used.[20] The target data set must be a PDS (not partitioned data set extended (PDSE)) with Fixed Block (FB) records. This command cannot change the 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 to EBCDIC conversion is provided.

The syntax is as follows:

```
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>]
```

[20] Special care is needed if ISV zPDT is in use. This command does not detect any serialization that z/OS uses and it is possible that the target data set member is also being used (and possibly edited) within z/OS.

- `ckd-file-name` is the Linux name of the file that contains the emulated volume.
- `PDS-name` is the z/OS name of the PDS.
- `mem-name` is a member name in the PDS.
- `Linux-file-name` specifies a Linux file to create (for extract) or write 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 that are used for EBCDIC or ASCII.
- `conversion`. 037/437 is the default, and 1047/437 might work better for international characters.

The PDS member name can be specified in any one of three ways. Using parentheses around the member name requires that the parentheses be escaped (so that the Linux shell does 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.)

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

We strongly suggest that you practice using **`pdsUtil`** on a test PDS before using it for a critical data set. Also, **`pdsUtil`** operates in your Linux system and has no knowledge of any RACF protection that might apply to the target PDS. This command is not intended for general usage. The design intention was for simple tasks such as changing an IP address or device name in a simple data set member.

Examples of usage are as follows:

`$ pdsUtil /z/WORK02 rb.admin.lib --list`
List the member names.

`$ pdsUtil /z/WORK02 rb.admin.lib/ICKDSF --extract`
Create `ICKDSF.txt`.

`$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --extract`
Create `ickdsf.txt`.

`$ gedit ickdsf.text#`
Use Linux editor.

`$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --overlay`
Because no Linux file was named, **`pdsUtil`** used `ickdsf.txt` instead.

`$ pdsUtil /z/WORK02 rb.admin.lib/ickdsf --overlay /tmp/myickdsf`
This example is valid but dangerous. The specified Linux file, `/tmp/myickdsf`, must be a valid overlay for the target member.

`$ pdsUtil /z/rb.admin.lib\(ickdsf\) --extract`
Must “escape” the parentheses.

`$ pdsUtil /z/rb/admin/lib --extract --mbr ickdsf`
Another way to specify how to extract a PDS member that is named `ickdsf`.

4.2.49 The query command

The **query** command displays the state of the CPs. ISV zPDT must be operational when using this command.

Here is the syntax of the command:

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

- **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 as follows:

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

An example of usage is as follows:

```
$ 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.

4.2.50 The query_license command

The **query_license** command displays the current ISV zPDT license status for Gen2 licenses, including the identity of the remote license server. ISV zPDT does not need to be operational to use this command, but the user must have configured a Gen2 client environment.

Here is the command syntax:

```
query_license
```

There are no operands. This command is not relevant for local token usage or for a remote Gen1 license server. This command can be used on both Gen2 clients and Gen2 servers. For a client, only information that is relevant to that client is displayed. For a server, more general information is displayed. This command is related to the Gen2 token and software licenses, and at the time of writing, the token and licenses are not available for ISV zPDT users.

4.2.51 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 ISV zPDT. Usually, this command is used when working with IBM (or an IBM Business Partner) while investigating a potential ISV zPDT problem. ISV zPDT does not need to be running when this command is used.

Here is the command syntax:

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

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

- **-s** indicates that only snap records will be displayed. At the time of writing, such records are not created by ISV zPDT releases, but might exist with older logs.
- **-t** indicates that records will be displayed in chronological order.
- **-d directory-name** overrides the normal logs directory name.
- **-c comp-name** indicates that only records about the indicated component will be displayed. This option is intended only for IBM internal use and is not further documented.
- **-u subcomp-name** indicates that only records about the indicated subcomponent will be displayed. This option is intended only for IBM internal use and is not further documented.
- **-b begin-time** indicates that only records after the indicated date and time will 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 that only records before the indicated date and time will be displayed. The format is the same as for **begin-time**.
- **-r rec-type** indicates that only the specified record type will be displayed. Valid types are `TRACE`, `LOG`, `LOG_REGBUF`, `QD_DUMP`, `LOG_EVENT`, `LOG_APPEND`, and `QUICK_DUMP`. Multiple operands can be separated with a comma.

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

- **$ rassummary** (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”**

4.2.52 The ready command

The **ready** command creates an “attention” or “ready” interrupt for the indicated device. It is most commonly used with an emulated tape drive to indicate that a new tape volume (which is actually a Linux file) was mounted or made ready. In some cases, **ready** can be useful with an emulated card reader or emulated local 3270 terminal. ISV zPDT must be running to use this command.

Here is the command syntax:

```
ready device-number
```

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

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

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

4.2.53 The request_license command

The **request_license** command creates a file that you send to an IBM authorized facility that generates Gen2 license files. This command is run with authority.

Here is the command syntax:

```
request_license
```

This command is related to the Gen2 token and software licenses, which at the time of writing are not available for ISV zPDT users.

4.2.54 The restart command

The **restart** command causes a PSW restart operation on the specified CP. ISV zPDT must be operational when using this command.

Here is the command syntax:

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

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

This command is seldom used. In some cases, it can be used to help an operating system that is stuck in an unusual situation, such as a restartable disabled wait. It also is used to dump a z/VM system and communicate with some stand-alone utilities.

The return values are as follows:

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

Examples of usage are as follows:

- **$ restart** (Restarts the default CP as set by the **cpu** command.)
- **$ restart 2**

4.2.55 The scsi2tape command

The **scsi2tape** command copies a tape volume that is mounted on a SCSI tape drive to a Linux file in awstape format. Linux files in awstape format can be managed and read by the awstape device manager as though they were tape volumes on a real tape drive. ISV zPDT does not need to be running to use this command.

Here is the command syntax:

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

- **-c** or **--compress** causes the output awstape file to be written in a compressed format. This command is not equivalent to hardware tape compaction, such as IDRC.
- **-i** or **--info** displays information about each tape file as it is processed. This operation is the default one.
- **-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, an IBM zSystems application requiring tape input does not know whether a “real” tape volume (on a SCSI tape drive) or an emulated tape volume (an awstape file on an emulated tape drive) is used. In practice, where repeated mounting and access to the tape might be needed, it might 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 an ISV zPDT operation. This format is not compatible with awstape formats on other platforms and 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`.

The return values are as follows:

`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 as follows:

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

4.2.56 The SecureUpdateUtility command

Important: This command was replaced with the **`Z1090_token_update`** or **`Z1091_token_update`** command for ISV zPDT release GA5 and later. Continue using **`SecureUpdateUtility`** for ISV zPDT releases before GA5.

The **`SecureUpdateUtility`** command is used to manage ISV zPDT license lease dates[21] in the ISV zPDT token. ISV zPDT must not be running when this command is used. (Linux warning messages are issued a month before the lease date in the token expires.) You must have root authority[22] and be in the `/usr/z1090/bin` directory before issuing this command.[23]

Here is the command syntax:

```
SecureUpdateUtility -r filename         (Use an arbitrary filename.)
SecureUpdateUtility -u filename.upw
```

The first form (**`-r`**) writes a *request* (`.req`) file for the token that is connected to the computer. Only one token should be connected when using this command. This request file is unique to the token that is connected. Some users (typically with 1091 tokens) do not need to create a request file. Your ISV zPDT supplier determines whether a request file is needed.

[21] Also known as the license expiration dates.
[22] The ISV zPDT program files are not normally in root’s PATH, so you might need to prefix the command with ./ (“dot slash”) to run it from `/usr/z1090/bin`.
[23] You can avoid using root by using the **`SecureUpdate_authority`** and **`zpdtSecureUpdate`** commands.

The second form (**-u**) applies the update file that is named in the command to the 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 is sent to a license processing facility that uses it to create the update file. Then, the update file is sent to the user, who applies it by running the **SecureUpdateUtility** command. An update file is unique to a token number and can be used only once. The license processing facility can return a `.upw` file, a `.zip` file, or both. You must have a `.upw` file to perform the update function with **SecureUpdateUtility**.

Examples of usage are as follows:

```
$ cd /usr/z1090/bin                  #(You must be in this directory.)
$ su                                 #(You must change to root.)
# SecureUpdateUtility -r myreq       #(Creates the myreq.req file in Linux. You
                                      #send the .req file for processing, and then
                                      #receive a .upw file in return.)
# SecureUpdateUtility -u myreq.upw   X(Apply the update file.)
# exit                               #(Exit from root.)
```

4.2.57 The SecureUpdate_authority command

The **SecureUpdate_authority** command adds a Linux user ID to or removes a Linux user ID from a set of user IDs that can issue the **zpdtSecureUpdate** command, which runs the **SecureUpdateUtility**, **Z1090_token_update**, or **Z1091_token_update** commands internally without requiring the user to operate as root and be positioned in the `/usr/z1090/bin` directory. With the **SecureUpdate_authority** and **zpdtSecureUpdate** commands, you can update token licenses without being root during an installation. The **SecureUpdate_authority** command must be run as root,[22] but it is typically used only once for a given user ID.

Here is the command syntax:

```
SecureUpdate_authority [-a | -d] userid
/usr/z1090/bin/SecureUpdate_authority [-a | -d] userid
```

- **-a** adds the indicated user ID to the list of user IDs that are allowed to issue the **SecureUpdateUtiluty** command.
- **-d** removes the indicated user ID from this list.

This command file is installed in `/usr/z1090/bin`, but this directory is not normally in the PATH for a root user. For this reason, you might use the full path name for the command (unless you are in `/usr/z1090/bin` when you issue the command). This command adds a record to the `/etc/sudo` file.

A command example is as follows:

```
# /usr/z1090/bin/SecureUpdate_authority -a ibmsys1 (Issued by root.)
```

Important: The **SecureUpdate_authority** command allows use of **sudo** and the **zpdtSecureUpdate** command permits token administration without requiring root authority. For more information, see 13.8, “Security exposures” on page 285.

4.2.58 The serverconfig command

The **serverconfig** command works with an Gen2 server, which must be installed before this command is used.

Here is the command syntax:

```
serverconfig [ ]                       (no operand)
             [-u | --update]
             [-r | --rules]
```

When the command is used with no operand, it displays a small interactive menu that is used to stop or start server operation or change log and UIM port numbers. The **--rules** option displays brief guidelines for the security file that is used by the LDK server. The **--update** option causes the server to reread the security file. The same functions that are performed by this command also can be performed by using the **serverconfig_cli** command, which is not interactive.

Root authority is not needed to start or stop the Gen2 server operation or logging function. This command is related to Gen2 token and software licenses, which at the time of writing are not available for ISV zPDT users.

4.2.59 The serverconfig_cli command

The **serverconfig_cli** command works with a Gen2 server, which must be installed before this command is used. You must be root to use this command.[24]

Here is the command syntax:

```
serverconfig_cli [-a[y|n]] [-l[y|n]] [-u]
```

The **-a** option enables or disables the Gen2 license server without stopping it. All connected ISV zPDT systems stop after a short interval. The **-l** option enables a server log. The **-u** option causes the server to reread the security rules in `/opt/IBM/LDK/rules`.

This command performs the same functions as the **serverconfig** command, but in a non-interactive manner. This command is related to Gen2 token and software licenses, which at the time of writing are not available for ISV zPDT users.

An example of the command is as follows:

```
# serverconfig_cli -ay -u
```

This example enables the Gen2 license server (which we assume was previously disabled) and rereads the security rules.

[24] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from `/usr/z1090/bin`.

4.2.60 The settod command

The **settod** command sets the specified time and date in the IBM zSystems time-of-day (TOD) clock during the next IPL of an active ISV zPDT system. The TOD change is not carried across restarts of ISV zPDT. When used, this command is normally issued after the **awsstart** command and before an **ipl** command. ISV zPDT normally sets the emulated IBM zSystems TOD clock to match the underlying PC TOD clock,[25] and this command alters that normal action. A **settod** command that is issued while an IBM zSystems operating system is active has no immediate effect, and takes effect only during a subsequent **ipl** command.

Here is the command syntax:

```
settod YYYY/MM/DD-HH:MM:SS
settod YYYY/MM/DD
settod HH:MM:SS                  (The :SS portion can be omitted.)
```

If both date and time are present, they must be separated with a dash (hyphen) 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 timestamps that are stored in an ISV zPDT token. This command provides a way to test IBM zSystems software at future times (or past times). After an IPL is done, the IBM zSystems TOD clock is incremented in the normal way, starting at the time and date that is specified in the **settod** command.

Assume the current date and time (in the PC hardware clock) is July 20, 2017 at 1 PM for the following example:

```
$ settod 16:40                   (July 20, 2017, 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. As a best practice, always enter a full date or time, or both.

[25] The leap seconds value can create a difference between the PC clock and the IBM zSystems TOD clock.

4.2.61 The st command

The **st** command is used to alter registers or memory in a CP, zIIP, zAAP, or IFL. ISV zPDT must be operational when using this command. The syntax is like the **d** command:

```
st {p xxx xxx xxx xxx  (expressed as four 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                                                }
   {vrm xxx      (z13; m=0..31)                           }
   {lso                                                   }
   {hex-addr xxx                                          }
```

- **xxx** is a hexadecimal value to be stored.
- **p** alters the current PSW.
- **pfx** alters the prefix register.
- **gn** alters the contents of a general-purpose register. A maximum of a 32-bit operand can be specified.
- **gxn** alters the contents of a general-purpose register. Up to 64 bits can be specified.
- **yn** alters the contents of a floating point register.
- **xn** or **xxn** alters a control register. The first format uses a 32-bit operand, and the second format uses a 64-bit operand.
- **zn** alters an access register.
- **vrm** alters vector register m.
- **lso** alters the leap second information block.
- **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 as follows:

0	Command is complete.
-2	No arguments are specified.

Command examples are as follows:

$ st p FF007AB0 0 0 123456	Set a 64-bit PSW.
$ st g2 123	Change the low-order 32 bits of GPR2 to x'00000123'.
$ st gx2 123	Change 64 bits of GPR2 to x'0000000000000123'.
$ st 461244 32	Change the byte at real address x'461244' to x'32'.

4.2.62 The start command

The **start** command starts a CP that was in the stopped state (due to an earlier **stop** command). ISV zPDT must be operational when using this command.

Here is the command syntax:

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

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

The return values are as follows:

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

Command examples are as follows:

- **$ start**
- **$ start 1**
- **$ start all**

4.2.63 The stop command

The **stop** command places a CP in the stopped state. The CP can be restarted with a **start** command or a reset function. ISV zPDT must be operational when using this command.

Here is the command syntax:

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

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

Generally, a CP is stopped 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 as follows:

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

Command examples are as follows:

- **$ stop**
- **$ stop 1**
- **$ stop all**

4.2.64 The storestatus command

The **storestatus** command causes certain CP registers to be stored in fixed memory locations, as defined in z/Architecture. ISV zPDT must be operational when using this command. This command might be used before taking a stand-alone dump (SAD).[26]

Here is the command syntax:

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

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

The CP must be in the stopped state (by using a **stop** command) when **storestatus** is issued. **storestatus** is no longer required before taking a SAD of z/OS.

An example of the command is as follows:

```
$ stop all
$ storestatus
```

4.2.65 The storestop command

This command existed in early releases of zPDT but is no longer available.

4.2.66 The stpserverstart command

The **stpserverstart** command is used to start the STP function daemons. The STP function is typically used only for a basic sysplex configuration. The `/usr/z1090/binwsCCT` file must be customized before this command is used. The same **stpserverstart** command is used on all Linux systems in the basic sysplex. It starts a server and client, or just a client, depending on the customization in your `/usr/z1090/bin/CCT_data` file. This command is not run from root.

Here is the command syntax:

```
stpserverstart
```

This command also adds the STP function to the **cron** lists of the Linux system, which causes the STP function to be restarted (if it fails) and automatically started when the Linux system is restarted. This command applies to all base systems (Linux systems) that are involved in the STP configuration. In an STP sense, some systems can be considered clients and others servers, but this command is used with *all* Linux base systems that are involved.

You must configure the `/usr/z1090/bin/CCT_data` file before starting an STP server or client.

4.2.67 The stpserverstop command

The **stpserverstop** command stops a running STP function. The **cron** entries that automatically start the STP function also are removed. This command is not used as root.

Here is the command syntax:

```
stpserverstop
```

[26] Recent z/OS and IBM zSystems usage removed the need for this command. IBM zSystems (including zPDT) is primed by z/OS to perform a **storestatus** command before the next IPL.

4.2.68 The stpserverquery command

The **stpserverquery** command displays the status of any STP functions running on your system.

Here is the command syntax:

```
stpserverquery
```

4.2.69 The sys_reset command

The **sys_reset** command performs a system reset (or system reset clear), as defined by z/Architecture. ISV zPDT must be operational when using this command.

Here is the command syntax:

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

- **normal** is the default operation.
- **clear** performs the additional clear function.

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

```
$ sys_reset
```

4.2.70 The tape2file command

The **tape2file** command reads a file from an emulated tape volume (in the 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. ISV zPDT does not need to be operational to use this command.

Here is the command syntax:

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

- **-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 the 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 data includes NL separators that indicate a Linux text file, but the separators are not required. IBM zSystems tape labels (in the input file) are not recognized, and they are treated as data files.

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

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

4.2.71 The tape2scsi command

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

Here is the command syntax:

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

- **-i** or **--info** displays information about each tape file as it is processed. This operation is the default one.
- **-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 the 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 that is used by ISV 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. For more information about tape drive naming, see Chapter 14, “Tape drives and tapes” on page 303.

Return values are as follows:

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 as follows:

```
$ tape2scsi -n  tape01.awstape /dev/st0
```

4.2.72 The tape2tape command

The **tape2tape** command copies a logical tape volume (in the awstape format) to another logical tape volume (also in the awstape format). Several optional operations can take place during the copy, including compressing or decompressing the data. The primary purpose of the command is to compress or uncompress awstape files or summarize a file. ISV zPDT does not need to be running to use this command.

Here is the command syntax:

```
tape2tape [-c        ][-d        ][-e nn   ][-i      ]
          [--compress][--dynainfo][--eof nn][--info  ]
                                            [-n      ]
                                            [--noinfo]

          [-s    ] in-file out-file
          [--scan]
```

- **`-c`** or **`--compress`** compresses the output tape.
- **`-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 parameter is the default.
- **`-n`** or **`--noinfo`** indicates no summary will be displayed.
- **`-s`** or **`--scan`** scans the tape, but no output is produced.
- **`in-file`** is the name of a Linux file in the awstape format.
- **`out-file`** is the name of a Linux file that is in the awstape format.

The input file can be in the ISV zPDT compressed format, which is handled automatically. The output file is compressed only if that option is selected. Both input and output files are in the awstape format. This command cannot convert other Linux files to the awstape format.

No return values are defined for this command. An example of the command is as follows:

```
$ tape2tape -e 1 /tmp/111111  /z/222222.awstape
```

4.2.73 The tapeCheck command

The **`tapeCheck`** command verifies the internal format of a logical tape volume in the awstape format, that is, it verifies that the awstape control bytes within the file are logically correct. ISV zPDT does not need to be running to use this command.

Here is the command syntax:

```
tapeCheck file-name
```

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

This command is used to inspect awstape files that might be corrupted. The command can be used to check awstape files that are generated on another platform to ensure that they are compatible with ISV zPDT. 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 that are found in the awstape format. An example of the command is as follows:

```
$ tapeCheck /tmp/222222.awstape
```

4.2.74 The tapePrint command

The **tapePrint** command writes the content of an emulated tape volume (in awstape format) to Linux stdout. ISV zPDT does not need to be running to use this command.

Here is the command syntax:

```
tapePrint [-a     ][-e      ] in-file
          [--ascii][--ebcdic]
```

- **-a** or **--ascii** specifies that the emulated tape volume has ASCII characters.
- **-e** or **--ebcdic** specifies that the emulated tape volume has EBCDIC characters, which is the default format.
- **in-file** specifies a Linux file that is in the awstape format.

The 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 as follows:

```
$ tapePrint /z/222222.awstape
```

4.2.75 The token command

The **token** command displays the characteristics of the ISV zPDT token that is in use. This command should be used when ISV zPDT is running.

Here is the command syntax:

```
token
```

There are no operands. Only the number of token licenses that are in use are displayed, that is, if the token allows three CPs but only one CP is in use, then information for only one CP is displayed. An example of the command usage follows:

```
$ token
CPU 0, zPDTA ... Serial 6186(0x182A) Lic=88570(0x159FA) EXP=4/15/2015 1090
```

The serial number is the effective IBM zSystems serial number, which might differ from the token serial number. The license serial number reflects the token serial number that is used to provide the ISV zPDT license. The final output indicator is 1090 or 1091, where 1090 indicates the ISV zPDT version and 1091 indicates IBM ZD&T.

The **token** output message can have “SHK” or “LDK” at the end. SHK indicates a Gen1 token or license manager, and LDK indicates a Gen2 token or license manager.

4.2.76 The txt2card command

The **txt2card** command reads a Linux text file and creates a card image file (in EBCDIC).

Here is the command syntax:

```
txt2card in-file out-file
```

- **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 to EBCDIC conversion table is fixed and cannot be customized.

There are no defined return values for this command. A command example is as follows:

```
$ txt2card /tmp/work2/config.txt /z/cards/deck1
```

4.2.77 The uimcheck command

The **`uimcheck`** command displays the state of the UIM serial number (which is the IBM zSystems serial number). ISV zPDT does not need to be running when this command is issued. Any user can issue the command.

Here is the command syntax:

```
uimcheck
```

There are no operands.

4.2.78 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 it is normally run while in the `/usr/z1090/bin` directory. This directory is not normally in the PATH for root. The recommended process is to change to root,[27] change to the `/usr/z1090/bin` directory, and use the command **`./uimreset x`**. The "./" bypasses the Linux PATH search and looks for the command in the current directory.

Here is the command syntax:

```
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 UIM function and its usage are explained in Chapter 8, "ISV IBM Z Program Development Tool licenses" on page 179.

4.2.79 The uimserverstart command

The **`uimserverstart`** command is used to start a remote UIM server. This command should always be issued by the same Linux user ID because it saves a database in the home directory of that Linux user. It must not be started by root.

Here is the command syntax:

```
uimserverstart
```

This command also adds the UIM server to the **`cron`** lists of the Linux system, which restarts the UIM server (if it fails). The UIM server automatically starts when the Linux system is restarted.

[27] An alternative is to use **`sudo`**.

A UIM server is normally used only as part of a remote ISV zPDT license server environment. It is not used when running a simple ISV zPDT environment with a token that is connected to the local ISV zPDT system. Do *not* use this command unless you are working on a Linux system running a remote UIM server.

The UIM function and its usage are explained in detail in Chapter 8, “ISV IBM Z Program Development Tool licenses” on page 179.

4.2.80 The uimserverstop command

The **`uimserverstop`** command stops a running UIM server. The **`cron`** entries that automatically start the UIM server are also removed. This command is not used as root.

Here is the command syntax:

```
uimserverstop
```

4.2.81 The update_license command

The **`update_license`** command installs a Gen2 software license file that was provided by an IBM authorized facility. This command must run with root authority.[28]

Here is the command syntax:

```
update_license
```

This command is related to Gen2 token and software licenses, which at the time of writing are not available for ISV zPDT users.

4.2.82 The Z1090_ADCD_install and Z1091_ADCD_install commands

The command that is appropriate for your Gen1 token (1090 or 1091) must be used. The description here is based on 1090 (ISV zPDT) tokens. The command is used to install the IPL volumes for the Application Development Controlled Distribution (ADCD) z/OS 2.1 (or later) release (and might apply to future z/VM ADCD releases). The command is used only for an IPL volume, and other volumes are installed by the Linux **`gunzip`** command. The command is normally not run by root. ISV zPDT *cannot* be running when this command is used, and if it is running, you see the `No license available` message.

You must have an appropriate ADCD license in your token or license server for this command to work.

Here is the command syntax:

```
Z1090_ADCD_install infile outfile
```

- **`infile`** is the distributed ADCD volume that is the IPL volume. (For recent ADCD z/OS systems, two volumes exist: the “normal” IPL volume and the recovery volume that is usually named SARES1.) These files normally have `.zPDT` as the file name suffix.
- **`outfile`** is the installed volume that is ready to use.

[28] The ISV zPDT program files are not normally in root’s PATH, so you might need to prefix the command with ./ (“dot slash”) to run it from `/usr/z1090/bin`.

The command requires more disk space in the directory that contains the outfile. This additional space is the same size as in the infile and used for a temporary work file that is automatically deleted before the command completes. The z/OS ADCD IPL volumes (with typical volume names such as xxRES1 and SARES1) are prepared differently for ISV zPDT and IBM ZD&T systems. For example, a 1090 (ISV zPDT) token cannot decrypt a SARES1 volume that is prepared for an IBM ZD&T customer.

Command examples are as follows:

- **`Z1090_ADCD_install /tmp/c5sys1.zPDT /z/C5SYS1`**
- **`Z1090_ADCD_install /tmp/sares1.zPDT /z/SARES1`**

4.2.83 The Z1090_token_update and Z1091_token_update commands

Important: This command might not be relevant if your ISV zPDT supplier provides token keys through a different method. For more information, contact your ISV zPDT supplier.

The command that is appropriate for your Gen1 token (1090 or 1091) must be used. The examples here are based on 1090 (ISV zPDT) tokens. This command is new with the ISV zPDT GA5 release and replaces the **`SecureUpdateUtility`** command.[29] You must be root and have `/usr/z1090/bin` as the current directory[30] to use this command.[31] ISV zPDT must not be operating when this command is used to update a token. You can use the **`-r`** function while ISV zPDT is active.

Here is the command syntax:

```
Z1090_token_update [-r filename[.req]]          (or Z1091_token_update)
                   [-u filename.[zip | upw]]
                   [-status]
```

- **`-r`** followed by an arbitrary file name creates a request file that is used to obtain token license updates.[32] The `.req` suffix is optional and automatically is added if it is not present. When the request file is sent to the licensing facility, you should receive a `.zip` file, a `.upw` file, or both in return.
- **`-u`** causes either the `.zip` or the `.upw` file to be installed. You can use the `.upw` file if you do not plan to install the ADCD z/OS 2.1 (or later) release. Use the `.zip` file if you plan to install this z/OS release.
- **`-status`** verifies that the token license files that are associated with ADCD usage are present.

This command obtains and installs updates for the ISV zPDT license and the licenses that are needed to install the ADCD z/OS 2.1 (or later) system. (Future z/VM releases might have similar requirements.) There is no harm in installing the token licenses that are needed for the ADCD z/OS 2.1 (or later) release even if you have no immediate plans to use that release. This command uses a small amount of temporary disk space in `/tmp`. When first installing the additional licenses (in the `.zip` file), this command can take longer than you might expect to complete.

[29] You should continue to use **`SecureUpdateCommand`** for ISV zPDT releases before GA5.

[30] The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from `/usr/z1090/bin`.

[31] You can avoid using root by using the **`SecureUpdate_authority`** and **`zpdtSecureUpdate`** commands.

[32] Some ISV zPDT users (typically with 1091 tokens) might not require a `.req` file to receive token updates. For more information, consult your ISV zPDT supplier.

Only one token can be present when this command is used. These commands are not used for Gen2 tokens.

An example command usage is as follows:

```
$ cd /usr/z1090/bin                  (You must be in this directory.)
$ su                                 (You must change to root.)
$ Z1090_token_update -r myreq        (Creates a myreq.req file in Linux. You
receive a .upw or .zip file, or both.)
$ Z1090_token_update -u myreq.zip    (Apply the update file.)
$ exit                               (Exit from root.)
```

4.2.84 The Z1090_removall command

The **`Z1090_removall`** command does exactly what the name implies: It removes ISV zPDT (including the two token drivers) from a Linux system.

Here is the command syntax:

```
z1090_removeall
```

This command requires root authority, and it is appropriate only in unusual circumstances.[33]

4.2.85 The z1090instcheck command

The **`z1090instcheck`** command checks a number of installation criteria. It can be used whether or not ISV zPDT is running. (The same **`z1090instcheck`** command is used for IBM ZD&T systems with a 1091 token).

Here is the command syntax:

```
z1090instcheck
```

There are no operands. This command is sensitive to the Linux distribution that is used and the level of that distribution. The output might vary with a new release of ISV zPDT. The **`z1090instcheck`** command is used with both 1090 and 1091 systems.

Return values are as follows:

0	Command completed.
8	An unrecognized Linux system is being used.

An example of usage is as follows:

```
$ 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
```

The specific report changes with new ISV zPDT releases and with the underlying Linux distribution. Some of the checks might produce warnings that you must evaluate for yourself. The help function (**`z1090instcheck -h`**) returns more configuration suggestions.

33 The ISV zPDT program files are not normally in root's PATH, so you might need to prefix the command with ./ ("dot slash") to run it from `/usr/z1090/bin`.

4.2.86 The z1090term command

The **z1090term** command provides an ASCII terminal function that can be used to connect to the HMC-like ASCII terminal that is defined by the **intASCIIport** statement in an ISV zPDT devmap. The syntax is as follows, where the IP address points to the base Linux system running ISV zPDT, and the port number is defined in the **intASCIIport** statement of the ISV zPDT devmap:

```
z1090term IPaddress:port
```

Examples are as follows:

```
$ 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 `/usr/z1090/bin` directory, and the executable file can be copied to other Linux systems where this terminal interface can be used to connect to ISV zPDT.

4.2.87 The ztrace command

This command controls the operation of an **address stop** function, that is, it causes zPDT to temporarily halt operation when an IBM zSystems system program runs an instruction within a specified virtual address range.

```
ztrace   [-a][--activate]                      (Activates tracing.)
         [-c CPUnumber | all][--cpu CPUnumber | all]        (Which CPU to trace)
         [-d][--deactivate]                    (Deactivates tracing.)
         [-i ASIDnumber][--id ASIDnumber]      (Which ASID to trace)
         [-l length][--length length]          (Length (in bytes) to monitor)
         [-p address][--pswa address]          (Address to start the trace)
         [-q][-query]                          (Displays the status of ztrace.)
         [-s][--stopall]                       (Stops all CPUs on a trace hit.)
         [-v][--verbose]                       (Verbose output display)
         [--usage]                             (Displays short ztrace usage message.)
         [-V][--version]                       (Displays ztrace version.)
```

Addresses, address space ID (ASID) numbers, and th length are specified in hexadecimal, but without the 0x prefix. The short or long names of the options can be intermixed, as shown in the rather complex example here:

```
$ ztrace --activate -c all -i 14 -p 1da1960 -l 10 -s
```

This command activates the trace function (**--activate**) on all emulated CPUs (**-c all**). The command monitors only ASID 0x14 (**-i 14**) starting address 0x1da1960 (**-p 1da1960**), extending for 0x10 bytes (**-l 10**), and stops all CPUs (**-s**) if an emulated IBM zSystems instruction in this address range runs. Then, the emulated CPUs can be started (to continue operation) with the zPDT command **start** or **start all**.

If an ASID number is not specified, then the **address stop** operation is global for all ASIDs and DAT modes. If an ASID is specified but a length value is not specified, then the **address stop** function is active in the whole address space. If the **stopall** operand is not specified, then only the CPU that triggered the stop is halted.

If the **activate** function is not specified, then the other specified operands are remembered, but the actual **address stop** monitoring is not operational. The monitoring can be started or restarted with a simple **ztrace -a** command.

Here is an example of the query function:

```
$ ztrace -q -c 1
   CPU: 1 state 800000000
      -Stopped
   ZTRACE PER IFETCH Enabled
   ZTRACE Event stops all CPUs
   Break Point Address: 0x1da1960
   Break Point Limit: 0x1da1970Address Space ID: 0x14
```

The **ztrace** operation can substantially impact zPDT performance. The address spaces and programs that are monitored are unaware of the **ztrace** function, and they can be expected to produce their normal results, assuming that the relevant **start** commands are issued as needed.

4.2.88 The z1090ver and z1091ver commands

The **z1090ver** or **z1091ver** command displays the current zPDT version, with the date that it was built. This information might be necessary when investigating a zPDT problem.

Here is the command syntax:

- **z1090ver** (for ISV zPDT)
- **z1091ver** (for ZD&T zPDT)

There are no operands. The return value is zero. An example of the command is as follows:

```
$ z1090ver
z1090, version z1090_v1r0_E39, build date - 10/17/08 SUSE 32 bit
```

The exact output messages vary with the zPDT release.

4.2.89 The zpdtSecureUpdate command

The **zpdtSecureUpdate** command automatically switches to the `/usr/z1090/bin` directory; runs the **SecureUpdateUtility**, **Z1090_token_update**, or **Z1091_token_update** command; and then switches back to your initial directory. You must be root to use this command[34] or you must be able to run this command through the **SecureUpdate_authority** command.

Here is the command syntax:

```
# zpdtSecureUpdate [-r | -u] Linux-file-name      (If you are root.)
$ sudo zPDTSecureUpdate [-r | -u] Linux-file-name  (If enabled for sudo.)
```

The operands (**-r**, **-u**, and **file-name**) are the same operands that are used with the **SecureUpdateUtility**, the **Z1090_token_update**, or the **Z1091_token_update** commands.

This command can be used in a manner that does not require root authority by the user. For more information, see 13.10, “Security exposures” on page 268.

[34] The ISV zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to run it from /usr/z1090/bin.

5

ISV IBM Z Program Development Tool installation

Important: This publication is about Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT). Many of the points that are described might also apply to IBM Z Development and Test Environment (IBM ZD&T) systems. If you are working with IBM ZD&T, you should obtain access to the full documentation that is provided for it.

Important: For ISV zPDT installation and operation, do *not* work with Linux root authority except when instructed to do so. This caution applies to both ISV zPDT installation and routine ISV zPDT operation.

Important: Some of the details that are mentioned in this publication (and especially in this chapter) are for ISV zPDT GA11, so they might differ from equivalent details for previous or later zPDT releases. GA11 was built on later Linux bases than previous ISV zPDT releases, which can change some of the Linux libraries that are needed.

ISV zPDT operates as a normal Linux application. As a best practice, follow the general procedures that are described here for initial ISV zPDT installation and usage. After you gain experience with ISV zPDT, you might explore other installation and usage arrangements.

This chapter describes the basic ISV zPDT installation process. Your product package might provide an enhanced installation method that is provided by your ISV zPDT vendor.

This publication does not provide detailed ordering information for ISV zPDT. The ordering process differs for various categories of users and for different countries. At a practical level, you must acquire the following items:

- A personal computer (PC) Linux operating system that is appropriate for your ISV zPDT release. This *base Linux* is not supplied by IBM. It must be ordered or downloaded from a vendor.
- An ISV zPDT token, which is known as an IBM 1090 token, and which *might* need to be activated through an IBM Business Partner, ISV zPDT supplier, or through IBM Resource Link®.
- The ISV zPDT software, which must be installed before the token can be activated (if it is not already activated). The base ISV zPDT software does not include any IBM zSystems operating systems.
- Whatever IBM zSystems software that you plan to use, in a format that is usable with ISV zPDT. This software *might* require a different ordering process than the one that you use to order ISV zPDT itself. Typically, the software is an Application Development Controlled Distribution (ADCD) package, as described in Chapter 6, “Application Development Controlled Distribution installation” on page 137. Much of the material in this publication assumes that you install the z/OS ADCD system. If you install different IBM zSystems software (and have an appropriate license for it), you must obtain specific instructions for ISV zPDT from the supplier of that software.

5.1 Linux installation

ISV zPDT GA11 was built and tested on several different Linux distributions, and informally tested on certain other distributions. For more information about which specific distributions were used, see 2.5, “ISV zPDT releases” on page 44. As a best practice, use one of the Linux distributions and levels that are listed. Earlier Linux levels might not work fully with ISV zPDT GA11, and later levels might contain subtle changes that create issues.

Disk planning

ISV zPDT does not care how you partition your Linux disks if you have sufficient room for whatever IBM zSystems emulated disk volumes that you need. A simple disk plan might be like the following one:

- A 30 GB partition, for example, for all the Linux files and the ISV zPDT program.
- An appropriate swap partition with 10 GB or whatever is appropriate.
- A large partition for emulated IBM zSystems volumes, which is *at least* 250 GB.

The separate Linux disk partition (and corresponding separate Linux file system) for the emulated IBM zSystems volumes allows the base Linux to be replaced easily without destroying the emulated volumes. In practice, most ISV zPDT customers have several disks, each with its own file system, and spread emulated volumes and various backup materials.

Basic adjustment example for installing ISV zPDT

Each of the various Linux distributions (and the distributions that are used for informal tests) might have a few “adjustments” that are needed for ISV zPDT installation and usage. Here is a specific set of additions that are needed by CentOS Stream 9 that was used for informal testing. The commands and syntax that are shown here were for that specific distribution, so slight changes might be needed with other Linux distributions.

We used the following commands after performing the basic CentOS Stream 9 installation from a USB flash drive in `.isv` format. These commands assume that your Linux system has an operational internet connection.

```
# yum install libstdc++.i686         (This library is needed for token
                                      operation.)
# yum install libnsl
# yum install perl
# yum install x3270                  (Optional. A 3270 emulator package.)
# yum install x3270-x11              (Needed by x3270.)
```

The details are likely to change slightly with other distributions. For example, the libstc++ library installation for SUSE might be **`zypper install libstdc++6-32bit`** and for Ubuntu might be **`apt-get install lib32stdc++6`**.[1] The library names here are the “search names” that you might use to find the library. Each library has a complete name, for example `libstdc++-11.3.1-2.1.el9.i686`.

During ISV zPDT installation, you might see error messages that indicate that other libraries are missing. If so, attempt to install them by using commands that are like the ones that are listed above. For example, for a SUSE installation, you might need **`zypper install libcap-progs`** or something similar.

ISV zPDT is not sensitive to the desktop manager. Some developers use GNOME and others use Xfce or KDE. A graphics Linux desktop is not required by zPDT, but many users find it convenient.

There might be other details that are involved in the initial Linux or ISV zPDT installation. For example:

- The ISV zPDT installation includes two Linux packages that are named `sntl-sud` and `shk-server`. You might find later versions of these packages elsewhere. Do *not* install the later versions. Use only the versions that are supplied with ISV zPDT.
- You might want the x3270 package. The package is probably not included with the Linux distributions, but it can be downloaded. Other 3270 emulation packages are available and might be used, but x3270 appears to be the most used one for ISV zPDT.
- We found that performing an online update for the Linux distribution is advisable when performing the initial Linux installation.
- You must manage whatever firewall and other security functions that you install with Linux. If possible, initially disable any firewall when first working with ISV zPDT. After you are familiar with the ISV zPDT operation, you can reestablish the firewall functions. If you have external TCP/IP connections (for local[2] 3270 connections, Open Systems Adapter (OSA) connections, license servers, Server Time Protocol (STP), or channel-to-channel (CTC) connections), you must provide the correct port *holes* in any firewall that you use.
- Select Coordinated Universal Time for your base PC, if possible. (This task might not be possible if you also run Microsoft Windows on the same PC.)

[1] These specific commands were correct in earlier releases of the indicated distributions. Later releases might need slightly different names.

[2] The definition of a “local” 3270 connection involves some historical background on IBM mainframes. If you are not familiar with the term, do not worry about it.

5.2 The ISV zPDT program package

The ISV zPDT program package is typically obtained by downloading it.[3] The package file contains the Red Hat, SUSE, and Ubuntu versions of the ISV zPDT code.[4] The correct version is automatically installed on your system.

5.3 ISV zPDT installation

Decide on the Linux user ID that you want to use for running ISV zPDT. We use `ibmsys1` in our examples, but you can select any user ID that is 8 characters or fewer (unless the **`system_name`** statement is used in the device map (devmap), in which case there is no special limit on the length of the Linux user ID). This publication frequently uses `ibmsys1` in examples.

In examples throughout this publication, the dollar sign prompt (`$`) indicates a non-root user ID, and the number sign prompt (`#`) indicates we are working as root. As a best practice, always log in as ibmsys1,[5] and then use a **`su`** command to switch to root when needed.[6] The following directions assume that a single ISV zPDT instance is used. (Multiple ISV zPDT instances require multiple user IDs, such as ibmsys2 and ibmsys3.)

If you have not done so, create user `ibmsys1` (or whatever user ID that you selected for ISV zPDT usage). By default, user ID `ibmsys1` uses `/home/ibmsys1` as its home directory, and some ISV zPDT control files appear in subdirectories there. We created file system `/z` as a separate partition during our Linux installation for our emulated volumes. We want user ID `ibmsys1` to own this file system.

To create `ibmsys1`, log on as ibmuser1 and run the following commands:

```
# su -                          (Switch to root.)
# chown ibmsys1 /z              (User ibmsys1 owns the emulated volume directory.)
```

A single executable file is used to install the ISV zPDT software. The file name changes with maintenance releases, but has the following general format:

```
z1090-1-11.57.05.x86_64 (Verify your exact file name.)
```

The single file contains the following items:

- An `sntl-sud` rpm at the correct level (a driver for the tokens).
- A `zpdt-shk-server` rpm at the correct level (another token program).
- The primary ISV zPDT rpm for SUSE Linux.
- The primary ISV zPDT rpm for Red Hat Linux.
- The primary ISV zPDT deb for Ubuntu.
- An *installer program* that displays a license and then installs the rpms or debs. The correct rpm or deb (Red Hat, SUSE, or Ubuntu) is automatically selected for your base Linux system.

[3] A different package, not generally covered in this publication, is used for IBM ZD&T systems. The proper package must be used with the correct token (1090 or 1091).
[4] The three ISV zPDT versions (in each package) are needed because the three Linux bases are typically at slightly different library levels or organizations.
[5] There is nothing special about user ID `ibmsys1`. It was a convenient choice.
[6] Another option is by using the **`sudo`** facility.

Place the ISV zPDT package file in a convenient library, such as `/tmp`. To do so, log in as `ibmsys1` and run the following commands:

```
$ su - (Change to root.)
# cd /tmp (If the file is in /tmp.)
# chmod u+x z1090-1-11.57.05.x86_64 (Make the file executable.)
```

Run the file, which runs as the installer program:

```
# ./z1090-1-10.57.05.x86_64 (First, verify the exact file name.)[7]
```

Scroll through the license information that is displayed and reply to the question at the end. Then, the various components are installed automatically. The ISV zPDT installer program performs the following tasks, removing existing versions of these programs as needed:

- Two prerequisite token modules are installed.
- The ISV zPDT rpm or deb is installed, mostly in `/usr/z1090/bin`.
- A set of `man` files is loaded into `/usr/z1090/man`.
- A `/usr/z1090/uim` directory is also created.

Installer options

If an ISV zPDT *fix* is installed, it *must* be removed before a new ISV zPDT release can be installed. (ISV zPDT fixes are rare, but are produced when needed for specific problems.) An ISV zPDT patch is an rpm or deb. Fixes can be found by running **`rpm -qa | grep z109`** commands and removed by running **`rpm -e`** commands, or their Ubuntu equivalents.

The installer program has three optional functions. Using the file name in the preceding example, the functions are listed here:

```
# ./z1090-1-10.57.05.x86_64 --refresh (Reinstall the ISV zPDT level.)
# ./z1090-1-10.57.05.x86_64 --refreshall (Reinstall ISV zPDT and its
                                          prerequisites.)
# ./z1090-1-10.57.05.x86_64 --removeall (Remove ISV zPDT and its prerequisites.)
```

The prerequisites that are mentioned here are two modules that are needed to access the USB token.

5.3.1 Altering Linux files

Important: You must complete the actions in this section before attempting to decrypt the initial program load (IPL) volumes of the z/OS ADCD system. You can use the supplied scripts (**`aws_sysctl`** and **`aws_bashrc`**) or edit the relevant files yourself, as described in this section.

You must alter two Linux files before you can use ISV zPDT. The first alteration is to `/etc/sysctl.conf`, which involves changing some Linux kernel parameters. The second alteration is to the `.bashrc` file in your home[8] directory, which adds the ISV zPDT directories to your user ID's PATH and LD_LIBARY_PATH variables. These changes are usually one-time changes. It is not necessary to make the changes again when upgrading to a new ISV zPDT release.

[7] The "./" characters before the file name tell Linux to run this file from the current directory.
[8] This home directory is the one for the Linux user ID that runs ISV zPDT. It is user ID `ibmsys1` in all our examples.

You can manually edit the relevant Linux files or use ISV zPDT commands to make the changes. The ISV zPDT commands are as follows:

```
# /usr/z1090/bin/aws_sysctl     (You must be root to use this command.)
$ /usr/z1090/bin/aws_bashrc     (You must not be root to use this command.)
```

The complete path name might be required for these commands (as shown here) because your Linux PATH might not yet include the ISV zPDT files. If you use these two commands, you can skip the following material about manually editing these files. You might want to restart Linux after making the changes.

Manually editing the files

If you want to manually edit /etc/sysctl.conf,[9] we suggest using **gedit**, but you can use any suitable editor (such as **vi** or **leafpad**) to add the indicated lines.[10]

Important: The **shmmax** and **shmall** values that are installed by the **/usr/z1090/bin/aws_sysctl** script are suitable for many laptop environments where the emulated IBM zSystems memory size is not more than about 24 GB. If your emulated IBM zSystems size is much larger, then you need larger **shmmax** and **shmall** values. Read the following notes carefully.

Some Linux distributions have acceptable values for **shmmax**, **msgmnb**, **msgmax**, and **core_uses_pid**, but other distributions might need to have all these values set.

```
# gedit /etc/sysctl.conf       (The following lines should begin in column 1.)
      kernel.core_pattern=core-%e-%p-%t
      kernel.core_uses_pid=1
      kernel.msgmax=65536
      kernel.msgmnb=65536
      kernel.msgmni=512                    (Change for large number of devices.)
      kernel.shmmax=36000000000
      kernel.shmall=24000000
      kernel.sem=250 32000 250 1024
      net.core.rmem_max=1048576
      net.core.rmem_default=1048576
# /sbin/sysctl -p /etc/sysctl.conf
```

Notes for sysctl values

The **shmmax** value establishes the maximum shared memory segment size that a user can request. All IBM zSystems memory, plus other ISV zPDT work areas, is in Linux shared memory. The **shmmax** value should be at least 20% larger than the IBM zSystems memory that is defined for the largest ISV zPDT instance that you use. The example that is shown (36,000,000,000 bytes) is suitable for an ISV zPDT instance with up to about 32 GB memory (as specified in the devmap). There is no need to attempt an exact fit for the **shmmax** number.

[9] Some versions of the IBM Open Client reset these values when maintenance is applied. If this scenario occurs, you should again enter the values that are shown here and run **/sbin/sysctl**.

[10] Do not attempt to use vi unless you have a basic familiarity with it.

Another parameter, **shmall**, sets the total shared memory size of all users. The value of **shmall** is specified in units of *page size*, which is usually 4096. The default value of **shmall** is usually large and acceptable.[11] However, if you have multiple ISV zPDT instances and all of them have large IBM zSystems memory, you might exceed the default **shmall** value. If this scenario happens, you must include a parameter like this one:

```
kernel.shmall=24000000
```

This parameter sets the total amount of shared memory, for all users, at 24,000,000*4096, or about 96 GB. This value should be greater than the number of ISV zPDT instances times the IBM zSystems memory size for each instance, plus about 10 - 20%.

Remember: The **shmmax** value is for the number of bytes, and the **shmall** value is for the number of pages.

The **kernel.msgmni** value, which is specified as 512 in this example, might need to be larger if you have many emulated I/O devices, for example, more than 150 devices. The **msgmax** and **msgmnb** changes are not needed for some Linux releases because they are the default settings. However, including these parameters in `sysctl.conf` appears to do no harm.

Included in the example is the **kernel.sem** parameter that controls the maximum semaphore configuration for Linux. If you have many devmap devices in multiple ISV zPDT instances, you might need to change this parameter. An example of a change is shown above. However, the default values are suitable for most users.

The **net.core** parameters might be needed if Ethernet large frames are used. These parameters seem to do no harm, so you can always include them. In this context, any frame with more than 1500 bytes is considered large.

Notes for .bashrc

The `.bashrc` file is changed, as follows:

```
# exit                              (Leave root if you are in root.)
$ cd /home/ibmsys1                  (My login directory)
$ gedit .bashrc                     (Use your favorite editor.)
   (Add the following lines beginning in column 1>):
      export PATH=/usr/z1090/bin:$PATH
      export LD_LIBRARY_PATH=/usr/z1090/bin:$LD_LIBRARY_PATH
      export MANPATH=/usr/z1090/man:$MANPATH
      ulimit -c unlimited
      ulimit -d unlimited
      ulimit -m unlimited      (If more than 128 emulated I/O devices)
      ulimit -v unlimited      (If more than 128 emulated I/O devices)
```

Double-check the entries in these two Linux files. Errors here might be difficult to detect later. The **ulimit -m** and **-v** statements are not required for most users and should probably be excluded unless you have more than 128 emulated I/O devices.)

[11] The default value on several distributions is 1,152,921,504,606,846,720, which is huge. However, on at least one Linux distribution, the default is much lower and might need to be adjusted.

5.3.2 Other Linux notes

Always use the same Linux user ID for ISV zPDT operation. This Linux user ID must be no longer than 8 characters unless the **system_name** statement is used in the devmap.[12] Multiple concurrent ISV zPDT instances require a different Linux user ID for each instance.

One of the last ISV zPDT installation steps that is shown above changes the **PATH** and **LD_LIBRARY_PATH** environmental variables that are used by Linux shells. Do not add other directories before /usr/z1090/bin in these variables.[13] There are many Linux commands that are provided with ISV zPDT, and they correspond to Linux file names that are accessed through the PATH variables. For example, the command **d** is used to display IBM zSystems memory and registers. If you place another directory containing a file that is named **d** earlier in the PATH, the ISV zPDT **d** function will not be available in the normal manner. Various internal ISV zPDT functions assume that they can access ISV zPDT modules through PATH and LD_LIBRARY_PATH, so you must ensure that this access is possible.

Restarting Linux

Restart Linux to pick up all the changes that you made. Then, use the **z1090instcheck** command to verify your environment for running ISV zPDT. Your new PATH is needed to find the command.

Log in as ibmsys1 and run the following command:

```
$ z1090instcheck
```

The same z1090 command is also used for 1091 systems.

If this command is not found, you do not have the PATH variables set or you did not install the ISV zPDT code correctly. This command does not check any devmaps that you might have defined or copied.

5.4 x3270

x3270 is an open-source Linux program that emulates IBM 3270 terminals.[14] It is not distributed with ISV zPDT, so you must obtain it yourself. You might find many different levels of x3270 on various websites, and some of them might not match the libraries that are available on your base Linux system. The three Linux distributions (Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server, and Ubuntu) that are used by ISV zPDT normally have a version of x3270 that is available in their repository.

To install x3270, run the following commands:

```
sudo apt-get install x3270
sudo apt-get install xfonts-x3270-misc  (This command might not be needed.)

yum install x3270
yum install x3270-x11                   (This command might not be needed.)
```

[12] The Linux user ID is used as the logical partition (LPAR) name under ISV zPDT, and LPAR names are limited to 8 characters or less. (Only a subset of LPAR-like functions is provided by ISV zPDT.)
[13] Directions for updating the PATH variables are given later in this chapter.
[14] There are variations of x3270 available, such as x3270c that might be used but are not discussed here.

Obtaining x3270 this way (from a Linux distribution repository) should provide a version that matches your base Linux libraries. Unfortunately, the x3270 versions that are provided this way are often earlier versions and might not have recent improvements. However, they provide the basic 3270 functions.

Many 3270 emulators are available for Microsoft Windows systems, and most of them can be used with local area network (LAN) connections to your ISV zPDT system.

5.4.1 x3270 keyboard maps

The default x3270 keyboard assignments are not in the traditional 3270 style. In particular, the large Enter key on the PC keyboard functions as the 3270 Enter key.[15] With traditional 3270 keyboards, this same key provides a *new line* function, and the 3270 Enter key is where the right-side Ctrl key is on most PC keyboards.

One way to change the key mapping is to create a file (in your home directory) that is named `.x3270pro` (note the period as the first character of the file name), which has the following content:

```
! Use Bill's overrides
x3270.keymap: bill
! Define the overrides
x3270.keymap.bill: #override \
   <Key>Control_R:  Enter()\n\
   <Key>Control_L:  Reset()\n\
   <Key>Return:     Newline()\n\
   <Key>Pause:      Clear()\n\
   <Key>End:        Clear()\n\
   <Key>BackSpace:  BackSpace() Delete()
```

The x3270 key map files are sensitive to extra spaces and tab characters. Do not have anything after the `\n\` in the text lines. In this file, we indented the `<Key>` field starting in column 4, although this action was arbitrary. Both the Pause and the End keys are mapped to the 3270 Clear function because some keyboards no longer have a Pause key.

Here are several default mappings for x3270:

```
PA1          alt-1
PA2          alt-2
F13          shift-F1              (and so forth for PF13-24)
```

x3270 fonts

If x3270 is installed from a separate rpm, it might not have its "normal" fonts.[16] In the x3270 fonts menu, there might be an option for ISO fonts. We selected the following one:

```
-eti-fixed-bold-r-normal--18-180-72-72-c-90-iso8859-1
```

The `18` that is embedded in the name is the point size. A similar choice, with `24` in this position, selected a larger font. Some x3270 versions produce error or warning messages about font mismatches when started, and these messages can be ignored if the 3270 panel function is acceptable.

[15] Users with strong z/VM backgrounds prefer this default key arrangement, and users with z/OS backgrounds (especially IBM Interactive System Productivity Facility (ISPF) usage) prefer to reassign the keys.

[16] There appears to be many levels or builds of x3270 on the web, and the differences include the way fonts are installed or used. We have experimented with different versions to find one that includes the wanted fonts.

5.5 Starting your new ISV zPDT system

For initial testing purposes, you might create the following file (which we arbitrarily name devmap0) in your home directory:

```
[system]
memory 8G
3270port 3270
processors 1

[manager]
name aws3274 1000
device 0700 3279 3274
```

Then, you can start ISV zPDT with the following command:

```
$ awsstart devmap0
```

You should see startup messages from ISV zPDT. If you have not yet installed an IBM zSystems operating system, there is not much else that you can do. You can stop ISV zPDT with the following Linux command:

```
$ awsstop
```

If these steps complete without errors, your ISV zPDT system is installed. If your token (or a connection to a remote license server) does not contain a valid license, you do not see a message saying `zPDTA license obtained`, but the startup and shutdown should work correctly.

For an initial ISV zPDT startup, install something like the above devmap example in your home directory. However, as a practical matter, we usually place our “real” devmaps (usually much larger than the above example) in the same directory as the emulated disk volumes. This action is not required, but appears to be a common choice of ISV zPDT users.

5.6 Installing a different ISV zPDT release

Important: New releases of ISV zPDT are typically at a new IBM zSystems architecture level. You must verify that your IBM zSystems software is suitable for the new architecture level. This verification might include the installation of “toleration” program temporary fixes (PTFs) or similar actions. Do *not* migrate to a new ISV zPDT release without considering your IBM zSystems software levels. The same considerations apply to installing a new large IBM zSystems machine.

New ISV zPDT releases are typically available through your IBM Business Partner or (for IBM employees) through IBM Resource Link. The installation procedure is the same regardless of the source. Installation is the same as described earlier.

A summary of the steps is as follows:

1. Obtain the new distribution file.
2. Working as root, run the distributed file. It deletes the previous release and installs the new release. The process takes only a few seconds and does not disturb your customization.

You might want to install an older ISV zPDT release for some reason, which is done in the same manner.

Application Development Controlled Distribution installation

The Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) program provides IBM zSystems functions and associated utility programs. It does not include any IBM zSystems software. IBM zSystems software, including operating systems, utilities, middleware, applications, and so forth, must be obtained separately. In practice, the IBM *product packages* for ISV zPDT typically include a z/OS Application Development Controlled Distribution (ADCD) package, although it is not part of the ISV zPDT program itself.

For software licensing purposes, an ISV zPDT system is an IBM zSystems machine, and all software licensing requirements that apply to a larger IBM zSystems installation also apply to an ISV zPDT installation. This statement applies to all IBM zSystems software from IBM, and we assume that is also applies to all IBM zSystems software that is available from other vendors.

The descriptions in this chapter assume that the correct licenses were obtained for the IBM zSystems software. Licensing arrangements can be complex topics and are not further addressed in this publication.

Important: The following description of the z/OS ADCD installation generally refers to the basic z/OS ADCD that is produced by IBM for ISVs.

Important: The descriptions in the remainder of this publication assume that the reader has a general familiarity with z/OS systems programming and understands how to access various control data sets. We highlight specific details that might be relevant to ISV zPDT usage and the current ADCD releases. This publication is not intended as an introduction to z/OS administration.

Furthermore, we assume basic familiarity with the ADCD z/OS package. You can find updated information about the ADCD packages at IBM zSystems Dallas ISV Center - Public.

6.1 General principles

At the time of writing, all IBM zSystems operating systems (assuming proper licenses exist) are supported for ISV zPDT usage, subject to architecture constraints, which includes current versions of z/OS and z/VM. Linux distributions that are intended for IBM zSystems usage might be used, but all possible functions and configurations have not been extensively tested. Older versions might work correctly if they are configured to work with the IBM zSystems architecture, which is the IBM z16 architecture in the case of zPDT GA11.

6.2 IBM zSystems operating systems

There are limitations for installing IBM operating systems or other software. These limitations are related to the usage of the software media and packaging techniques that are involved, and are not limitations on the usage of the software after it is installed.

A common limitation is for any software that is distributed on tape. To install this software, your ISV zPDT system must have a tape drive, which is not commonly available for personal computer (PC) machines. Another limitation is related to any IBM zSystems software that is packaged in such a way that installation requires specific IBM zSystems Hardware Management Console (HMC) functions.

6.2.1 Media

In most cases (when a tape drive is not available), installation media is limited to DVD or local area network (LAN) connections. (You can consider FTP or another form of downloading as *media* in this context.) Distributed files must be in formats that can be processed for ISV zPDT. There are two meaningful formats:

- A Linux image of an emulated 3390 drive[1] that can be restored in the 3390 format that is used by ISV zPDT. The image might be compressed (by using **`gzip`**, for example) and must be decompressed before using it with ISV zPDT. Likewise, the image might be in `.tar` files and must be extracted (and possibly decompressed) before being used with ISV zPDT.

[1] 3380 images also can be used, but we ignore them here.

The 3390 drive image format must be produced by another zPDT system because no other product uses the same 3390 image format that is used by ISV zPDT. Whatever preliminary unpacking or decompression is needed must be done by Linux utilities before the 3390 image can be used by an IBM zSystems operating system running under ISV zPDT.

- A tape image in awstape format. Such images appear as “real” tape volumes to IBM zSystems operating systems on ISV zPDT, and they can be processed as such by using emulated tape drives. The tape might contain product installation material (in SMP/E format, for example), an ADRDSSU dump of a disk volume, or any other tape data that is usable by IBM zSystems programs.

Another media option is to use FTP to download a product (or other data) directly to z/OS. Some IBM zSystems software is distributed in this format. ISV zPDT must have a running z/OS and LAN connection to use this method. Most of our description is for z/OS, but z/VM or Linux for IBM zSystems might be used in the same way. The point is that a working IBM zSystems operating system must be installed before more software can be sent directly to it through FTP.

Differentiating the handling of these methods is important:

- DVDs must be processed by Linux programs (unless they contain awstape files).
- The awstape files must be processed by z/OS (or another IBM zSystems operating system), although the transport of awstape files can be managed by Linux through CD/DVDs, USB memory keys, FTP, or another method.
- Files that are sent through FTP directly to z/OS might be in other formats, for example, in formats that are suitable for processing by SMP/E or the Time Sharing Option (TSO) **`XMIT`** command. The formats are IBM zSystems formats and not Linux formats.
- An emulated 3390 volume (after decompression, if necessary) is a large Linux file that is meaningful only to IBM zSystems software.

ADCD disk sizes

When first inspecting an ADCD z/OS download (assuming that you have an authorized access to the IBM site that is involved), you might wonder about the disk sizes that are involved. For example, your initial inspection might find a `b5cfg1.gz` volume[2] that appears to be about 160 MB. At the time of writing, almost all ADCD z/OS volumes are, in effect, 3390-9 volumes that (when uncompressed) occupy about 8.5 GB of disk space. The compressed state is useful for distribution but cannot be used for z/OS operation under ISV zPDT.

The number of volumes that are needed for z/OS (for the release at the time of writing) varies with whether functions such as CICS or IBM Information Management System (IMS) or the basic DLIBs are selected, but typically starts with about 15 volumes (15 * 8.5 or about 130 GB of disk space for the basic z/OS volumes). Volumes for your local data or other options probably add to this number. Some of the ADCD z/OS volumes are almost full while others are mostly empty, but the volumes still exist.

[2] This particular volume name (c4cfg1) is associated with a particular level of the ADCD z/OS 2.4 system. The exact name varies with other levels, and the concept is the important part of this description.

6.3 Installing a z/OS ADCD system

The following examples use volsers (IBM zSystems volume serial numbers) that correspond to the z/OS 2.5 release of the ADCD system. These volsers tend to change in a standard pattern for new releases.

The initial program load (IPL) volumes of z/OS ADCD releases are encrypted. The two volumes that can undergo an IPL are usually xxRES1 and SARES1, where “xx” changes with each z/OS ADCD update. These volumes must be installed by using the **`Z1090_ADCD_install`** command. This command decrypts the volumes and helps implement a signature technique that identifies the customer installing the volume.[3]

6.3.1 Specific installation instructions

There are multiple download files for z/OS ADCD releases, with one file for each emulated 3390 volume. Documentation with each ADCD release contains specific information about the volumes that are needed for that release.

Volumes other than IPL volumes are all in the **`gzip`** format. ADCD system installation might be as follows, assuming your target directory for emulated 3390 volumes is `/z` (we assume that you are working as user ID *ibmsys1*):

```
$ cd /tmp                             (or wherever you stored the downloads)
$ Z1090_ADCD_install b5res1.zPDT /z/B5RES1          (Decrypt IPL volume.)
$ gunzip -c b5res2.gz > /z/B5RES2                   (Decompress other volumes.)
$ gunzip -c b5uss1.gz > /z/B5USS1
$ gunzip -c b5sys1.gz > /z/B5SYS1
```

And so forth, for all the volumes to be installed. Use the **`Z1091_ADCD_install`** command instead of **`Z1090_ADCD_install`** if appropriate.

The suffix of the file name for a distributed IPL volume is `.zPDT` or `.ZPDT` instead of `.gz`. The distributed file is encrypted and compressed, and it is automatically expanded as part of the **`Z1090_ADCD_install`** processing.

The files containing emulated volumes (and the directory containing these files) must have read/write permissions for the user ID running ISV zPDT. Assuming that you use the `ibmsys1` user ID, we suggest that all such files and their directories (`/z` in our examples) be owned by `ibmsys1`.

File name considerations

An emulated 3390 volume exists in a single Linux file. For example, a 3390-9 volume exists as a 8.5 GB Linux file. A 3390 volume has a volser that is written in the first track of the 3390 volume.[4] The Linux file holding the (emulated) 3390 volume has a Linux file name, which is specified in a devmap. There is no required relationship between the volser and the Linux file name. For example, volser WORK02 might be in `/tmp/mysys/ckd001`.

[3] z/OS ADCD releases before z/OS 2.1 were not encrypted. z/OS ADCD releases starting with z/OS 2.1 have encrypted IPL volumes. To decrypt these volumes, you must have ISV zPDT GA5 or later, and you must have a token license file that contains the proper license to enable decryption. The proper license files are distributed in `.zip` format. If you have earlier software (before z/OS 2.1 or before ISV zPDT GA5), you should refer to earlier editions of this publication.

[4] The volser is written by the ICKDSF utility program (for a new volume), or is present in a volume that was downloaded or taken from a DVD.

In our examples, we elected to use the volser of the 3390 volume as the Linux file name that holds the volume. We use uppercase letters to make these emulated volume file names more distinctive. There is no requirement to use the volser as the Linux file name, and there is no requirement to use uppercase names.

As a best practice, make the Linux file name reflect the volser, if at all possible. For example, volser WORK02 might be in `/z/mysys/WORK02` or `/z/mysys/work02.ckd`. A Linux naming convention that reflects the volser can avoid wasted time.

6.3.2 Input/output definition file device numbers

You must know the device numbers (commonly known as addresses) that are used by the z/OS system. (These addresses can be changed after the z/OS system is installed. Changing these addresses involves creating an input/output definition file (IODF) data set and IPLPARM member or members, and then performing an IPL of z/OS again.[5]) Most users of the ADCD system accept the device numbers in the IODF that is supplied with the ADCD system, which are listed in Table 6-1.

Table 6-1 IODF device numbers for ADCD

Address	Device	Purpose
00C	2540R	Card reader. Useful as an emulated device.
00E - 00F	1403-N1	Line printers. Useful as an emulated device.
120 - 15F	3380	Disks. (Control units that are defined for 120 - 127.)
300 - 318	3390	3390 disks.
400 - 40F	Open Systems Adapter (OSA)	OSA.
550 - 55F	3420	Round tape drives.
560 - 56F	3480	Without the COMPACT feature.
580 - 58F	3490	Tape drives.
590 - 59F	3590	Tape drives.
600 - 60F	3990	Disks.
700	3270	Terminal. ADCD systems use it as the NIP and z/OS master console.
701 - 73F	3277	Terminal. Normally for IBM Virtual Telecommunications Access Method (VTAM) (TSO, CICS, and so on).
900 - 910	3277	Terminal. Normally for VTAM (TSO, CICS, and so on).
908	3270	Could be used as a z/OS console.

[5] Creating a IODF is not a typical task for application programmers. We suggest using the supplied IODF unless you have experience in this area.

Address	Device	Purpose
909 - 91F	3277	Terminal. Normally for VTAM (TSO, CICS, and so on).
A80 - AFF	3390	Disks.
E20 - E23	CTC	3172 or CTC devices.
E40 - E43	CTC	3172 or CTC devices.
1A00 - 1AFF	3390	Disks.
2A00 - 2AFF	3390	Disks.
3A00 - 3AFF	3390	Disks.

Most of the addresses are 3 hex digits, due to historical reasons. Both the ADCD z/OS system and ISV zPDT system can work with 4-digit addresses. These addresses have been stable for many releases of the z/OS ADCD system; however, it is possible they might change in future releases.

In principle, the 3390 IPL volume, for example, can be mounted at any address that is defined as a 3390. By ADCD convention, the IPL volume and the volume containing the IODF and other key data sets are usually mounted at addresses A80 and A82, as shown in Table 6-2.

Table 6-2 IODF and other key data sets mounted at A80 and A82

Volser	Address	Purpose
B5RES1	A80	IPL volume and key z/OS libraries
B5SYS1	A82	Spool space, LOGGER data sets, Virtual Storage Access Method (VSAM), and so on

The A80 and A82 addresses are used in ADCD documentation examples, but there is no requirement to use specific addresses. The other volumes can be mounted at any convenient address that is defined in the IODF as a 3390. In practice, many users start at address A80 and increment it sequentially for each additional volume.

6.3.3 ISV zPDT control files

Before the ADCD system can be used, an appropriate devmap must be created. A rather basic example is shown here:

```
$ cd /home/ibmsys1
$ gedit aprof23                    (An arbitrary file name)
   [system]
   memory 9000m                    # Define a 9000 MB IBM zSystems system.
   processors 1                    # Use 2 or 3 if appropriate.
   3270port 3270                   # The port number for TN3270 connections

   [manager]
   name aws3274 0002               # Define a few 3270 terminals.
   device 0700 3279 3274
   device 0701 3279 3274
   device 0702 3279 3274
   device 0703 3279 3274
   device 0704 3279 3274

   [manager]
   name awsckd 0001
   device 0a80 3390 3990 /z/B5RES1        # (The “B5” prefix is for the
   device 0a81 3390 3990 /z/B5RES2        # z/OS ADCD release.
   device 0a82 3390 3990 /z/B5SYS1        # Later releases will have a
   device 0a83 3390 3990 /z/B5CFG1        # different prefix.)
   device 0a84 3390 3990 /z/B5USS1
   device 0a85 3390 3990 /z/B5USS2
   device 0a86 3390 3990 /z/B5PRD1
   device 0a87 3390 3990 /z/B5PRD2
   device 0a88 3390 3990 /z/B5PRD3
   device 0a89 3390 3990 /z/B5PRD4
   device 0a89 3390 3990 /z/B5PAGA
   device 0a8a 3390 3990 /z/B5PAGB
   device 0a8b 3390 3990 /z/B5PAGC
   device 0a8c 3390 3990 /z/B5USR1
   #(Continue with whatever additional volumes that you installed.)
```

Gaps in the assigned address numbers do not create a problem. The devmap can have any name and be placed in any directory. It is best if it is in the directory that you use when starting ISV zPDT so that you do not need to enter a full path name when using it.

We suggest that you do not define OSA devices for your initial z/OS startup. The OSA definitions can be a little more complex, and we suggest that you verify that your basic z/OS system is operational first.

6.3.4 IPL and operation

Start ISV zPDT with the **awsstart** command. Among other functions, this command starts the ISV zPDT device manager that emulates local, channel-attached 3270 terminals. Using the **awsstart** command creates a z1090 subdirectory in the current home directory (if it does not exist) and several ISV zPDT-related directories below it, as shown in the following string:

```
$ cd /home/ibmsys1
$ awsstart aprof2                    (Use your devmap name. Wait for messages.
                                      Press Enter to regain the $ prompt.)
AWSSTA014I Map file name specified: aprof22
0 Snapdump incidents, RAS trace and RAS log files occupy 657046 bytes
in /home/ibmsys1/z1090/logs.
Associated files, logs, and core files occupy 10364 bytes in
/home/ibmsys1/z1090/logs
```

Using the same Linux window (or a different window, if you prefer), start at least two local 3270 sessions:[6]

```
$ x3270 -port 3270 localhost &
$ x3270 -port 3270 localhost &
$ x3270 localhost:3270 &          (another way to specify a port number)
```

[6] These instructions assume that you installed x3270 and are using it on the same PC that is running ISV zPDT.

Consider the following information:

- `x3270` is the name of the program.[7]
- In the devmap, we assigned Linux TCP/IP port 3270 for this function. The port number is arbitrary, but should not be used for any other purpose in your system. Port 3270 is usually a good choice and is easy to remember.
- We want to connect to our own Linux system, which is indicated by the `localhost` operand.
- The ampersand (`&`) causes the x3270 program to run in the background, leaving the Linux window free for more commands. You can recall and run the **`x3270`** command repeatedly to create multiple 3270 sessions.

The 3270 window displays identification lines if no data has been sent to it by the IBM zSystems system software. These lines indicate the terminal identity by address and LUname or IP address. A number of options are available for working with these LUnames, which are described in 3.6.3, “The aws3274 device manager” on page 59. The **File** and **Options** menus at the top of the x3270 window can be used for various functions. Changing the font size (by using the **Options** menu) changes the 3270 window size.

The 3270 session for the z/OS operator console (address 700 for the ADCD system) should be ready before the next z/OS IPL. Then, issue the appropriate IPL command in the Linux window:[8]

```
$ ipl a80 parm 0a82cs
```

After a few seconds, the initial z/OS messages appear on the 3270 session at address 700. During the first IPL of the ADCD system (or an IPL after a long period of non-use, or a changed IBM zSystems serial number), you might see messages like this one:

```
IXC420D REPLY I TO INITIALIZE SYSPLEX ADCDPL, OR R TO REINITIALIZE XCF
```

If this message is issued, go to the 3270 session that shows the message and enter the following command:

```
r 00,i
```

After VTAM is started, the VTAM logo should appear on the other 3270 sessions.[9]

There is usually some documentation for each ADCD release that provides details about different IPL parameters and TSO logon procedures. A brief summary for a recent z/OS ADCD system is shown in Table 6-3.

Table 6-3 Summary for a recent z/OS ADCD system

IPLparm	LogonProcedure	Purpose
0A82CS	ISPFPROC	Basic IPL without Db2. Cold start JES2.
0A8200	ISPFPROC	Subsequent basic IPLs. Warm start JES2
0A82DC	DBSPROCC	Initial IPL for Db2 V12.

[7] Assuming that you installed the open source x3270 program, which is not included with ISV zPDT.

[8] This example assumes that you mounted the IPL volume at address A80 and the volume containing the IODF and IPL parameters at address A82. By ADCD convention, the “cs” IPL parameter causes a Job Entry Subsystem (JES) 2 cold start.

[9] If the 3270 session displays the message `Unsupported Function`, use the 3270 Clear key to obtain the initial VTAM display. Some TN3270e emulators encounter this initial message and others do not.

User IBMUSER is present on z/OS and typically used for initial TSO logons. The initial password for IBMUSER should be published with the ADCD documentation. It is typically SYS1 or IBMUSER. If there are security concerns about your system, change this initial password as soon as possible.

The distributed logon procedures change with various ADCD releases. The procedures are in the ADCD PROCLIB data set.

The Linux command-line interface (CLI) that was used for the **awsstart** command should be kept open, if possible. Asynchronous messages from ISV zPDT are sent to this window. You can enter ISV zPDT commands from other windows, but it is possible that you might miss significant messages that are sent to the original window.

Important: Our examples use “cs” in the “parm” field. In the ADCD z/OS systems, “cs” usually indicates that the JES2 spool should be cleared, that is, any existing material should be erased. You probably do not want to do this task for your routine IPLs. In those cases, you might use “00” instead of “cs”, although there are other options that are available. The “00” (as used in the default ADCD configurations) indicates a basic z/OS IPL that does not clear existing content in JES2.

6.3.5 Shutting down

z/OS should be shut down cleanly, if possible. Enter the **s shutdown**[10] command at the z/OS console and reply to any messages that are produced. There is typically a considerable pause before the message ALL FUNCTIONS COMPLETE indicates that JES2 can now be stopped with the command **$PJES2**. After JES2 ends, the IBM zSystems system operation can be stopped.[11] The ISV zPDT system is stopped with the following command in the Linux window:

```
$ awsstop
```

This command produces several messages. It might be necessary to press Enter to obtain the Linux prompt. Any 3270 windows can be closed now.

6.3.6 Startup messages

Messages such as the following are produced by the **awsstart** command:

```
AWSSTA014I Map file name specified: aprofa2
0 Snapdump incidents, RAS trace and RAS log files occupy 657046 bytes
in /home/ibmsys1/z1090/logs
Associated files, logs, and core files occupy 10364 bytes in
/home/ibmsys1/z1090/logs
```

Look at these messages because SNAP dump incidents are indications of an internal ISV zPDT error, and if you want to work with your ISV zPDT support, you might need this data. The number of bytes that is used for various logs and dumps is usually not significant unless it becomes too large. If the numbers that are displayed become too large (many megabytes) and if you are not actively working on a problem with your ISV zPDT support organization, you might want to clean these files. To do so, add the **--clean** option the next time you issue an **awsstart** command:

```
$ awsstart aprof11 --clean
```

[10] This **shutdown** procedure is not a standard z/OS function because additional command names might be present. These additional commands trigger VTAMAPPL scripts to issue various commands that are involved in stopping z/OS.

[11] Many users issue a z/OS **quiesce** command at this point.

You can get the **--clean** behavior every time by setting the Linux shell environment variable `Z1090_CLEAN=YES`; however, we do not suggest doing so because it might easily result in the removal of important debugging information in an ISV zPDT failure.

6.3.7 Copying ADCD gz files

Beware of a special issue when trying to copy ADCD `gz` files to a USB flash drive or a DVD. For more information, see “Copying large Linux files” on page 278.

6.3.8 Local volumes

For our examples, we created a Linux disk partition that is mounted at `/z`. To add your own 3390 volumes, complete the following steps:

1. Create the emulated 3390 volume by using an ISV zPDT utility:

   ```
   $ alcckd /z/WORK01 -d3390-3          (assuming that you want a 3390-3 volume)
   ```

2. Update the devmap to include the new volume. (Assume address AA0 for this example.)

   ```
   [manager]
   name awsckd 0001
   ...
   device AA0 3390 3990 /z/WORK01
   ```

3. Restart ISV zPDT with the updated devmap.
4. Perform an IPL of z/OS with the new volume present. z/OS detects an uninitialized volume and automatically varies it offline.
5. Create and run an ICKDSF job to initialize the volume:

   ```
   //BILLX JOB 1,OGDEN,MSGCLASS=X,REGION=40M
   // EXEC PGM=ICKDSF,REGION=0M
   //SYSPRINT DD SYSOUT=*
   //SYSIN DD *
    INIT UNIT(AA0) NOVERIFY VOLID(WORK01) VTOC(0,1,14)
   /*
   ```

6. Vary the new volume online and begin using it:

   ```
   VARY AA0,ONLINE                 (on the z/OS console)
   ```

6.4 Multiple operating systems

You can install multiple IBM zSystems operating systems. You are limited only by the disk space that is available. Every emulated 3390-3 volume uses approximately 2.8 GB of disk space, and 3390-9 volumes use about 8.6 GB. (Various other sizes can be used.)

It is important to distinguish between *installing* more emulated 3390 volumes (perhaps with various operating systems) and *using* the volumes. You can perform an IPL of only a single system at any one time in an ISV zPDT instance.[12]

[12] This statement ignores the possibility of running multiple z/OS guests under z/VM.

The volumes that might be “seen” by that system depend on several factors:

- Does the current devmap contain all the wanted volumes?

  You can have multiple devmaps, each with a different selection of emulated volumes and assigned addresses, but you can have only one devmap that is specified when you start an ISV zPDT instance. You cannot change the active devmap while ISV zPDT is running.[13] (You can change the devmap file after IDV zPDT is started, but this file change has no effect on the running ISV zPDT.)

- Do the device addresses in the devmap match suitable addresses in the IODF of the z/OS system?

  For example, if one of the emulated 3390 volumes is assigned address 190 (in the devmap), then the default z/OS ADCD IODF cannot “see” the volume because this address is not in the IODF. (z/VM does not have predefined addresses for various device types, making this aspect of z/VM easier to use.)

- Duplicate disk volsers should not be present.

  You can have duplicate volsers for emulated volumes on your PC disk, but the duplicates should not be present in a given devmap.

- It might not be possible to use the common addresses that typically are associated with an operating system.

  For example, all the ADCD documentation examples use A80 as the IPL address for a z/OS ADCD system. You can have two (or more) ADCD systems that are represented in the devmap concurrently, but only one volume can have address A80 during any single execution of ISV zPDT. This situation does not prevent you from performing an IPL of any of the (multiple) ADCD systems that are present, but you must specify the correct address. For example:

  ```
  Address  VOLSER   Purpose
  A80      C4RES1   IPL volume for z/OS 2.4 ADCD system
  A81      C4RES2   Libraries for z/OS 2.4 ADCD system
  A82      C4SYS1   Paging, spooling, VSAM for 2.4 ADCD system
  ...
  A90      D3RES1   IPL volume for z/OS 2.3 ADCD system
  A91      D3RES2   Libraries for z/OS 2.3 ADCD system
  A92      D3SYS1   Paging, spooling, VSAM for 2.3 ADCD system
  ...
  ```

  Assuming that the devmap is configured for these addresses, you can run **`ipl A80 parm 0A82CS`** to run the 2.4 system or **`ipl A90 parm 0A92CS`** to run the 2.3 system.[14] In either case, the running z/OS system can access all the volumes of both z/OS systems, which is convenient for migration purposes.

You can run multiple IBM zSystems operating systems concurrently by starting multiple ISV zPDT instances, but this action requires more resources (especially PC memory).

[13] This statement is not completely true. You can change the volume that is mounted on an emulated disk drive or tape drive with the **`awsmount`** command.

[14] Yes, we know that these levels of z/OS are ancient and do not work on the current ISV zPDT system, but they illustrate the point being made.

7

Local area networks

A local area network (LAN) setup can become complicated, partly because you must deal with physical LAN hardware and interfaces in the underlying Linux, IBM zSystems operating system configuration and definitions, and possibly with external routing complexity. At the IBM zSystems level, Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) provides emulated Open Systems Adapter (OSA) functions plus something like an emulated IBM 3274 control unit connection. OSA functions are closely tied to CHPID usage (on a large IBM zSystems system), and you must deal with CHPID-like details at the ISV zPDT level. An ISV zPDT system can use more than one personal computer (PC) LAN adapter, although that is unusual. The following descriptions are based on z/OS usage, but similar considerations are present with other IBM zSystems operating systems.

In this chapter, we consider only Ethernet connections. We use “LAN” and “Ethernet” interchangeably, and consider these terms to include wired or wireless Ethernet and Ethernet “tunnel” operation, where appropriate.

Attention: At the time of writing, ISV zPDT provides OSA operation in both QDIO and non-QDIO modes. As a best practice, use QDIO mode unless you have a specific need for non-QDIO operation. Appendix B, “Non-QDIO Open Systems Adapter” on page 375 provides information about non-QDIO operation. Non-QDIO operation is not further considered in the current chapter.

For our purposes, a wireless connection is treated as another Ethernet adapter, and it can be used as such by Linux TCP/IP and OSA emulation. Generally, wireless connections are slower than wired connections, which might be relevant for large FTP transfers. Also, wireless connections are more prone to disruptions than wired connections, which can cause problems if, for example, the link to the MVS console is disrupted.

Our examples include basic usage of Linux **`ip`** commands and **`firewall-cmd`** commands. You might need more complex usage of these commands (or other network commands), but we do not attempt to describe that usage.[1]

[1] There appears to be a progression of “network” management commands in Linux that seems to vary somewhat between different Linux distributors and updates.

Attention: The x3270 program (mentioned frequently in this publication) is primarily supported by Paul Mattes. Paul does this support in the public domain, and he is not an IBM employee. Also involved with the program are Don Russell, Jeff Sparkes, Dick Altenbern, and GTRC.

7.1 Overview of LAN usage

Several key factors are described in this chapter:

- A PC Ethernet adapter can be used concurrently by both OSA 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 z/OS (through ISV zPDT OSA) is using the same Ethernet adapter. This configuration is a common one for ISV zPDT.
- You do not need a z/OS TCP/IP operation for one key mode of 3270 access to z/OS. Access can be through the aws3274 device manager, and it appears as local, channel-attached 3270 terminals to z/OS. OSA definitions are not used. This scenario is our first one.
- z/OS and the base Linux cannot communicate with each other through the same hardware LAN adapter. Both can share the hardware LAN adapter for all TCP/IP functions *except* communicating with each other. ISV zPDT implements a *tunnel*[2] pseudo-LAN to bypass this restriction.
- Standard z/OS is not a DHCP client. You cannot plug in z/OS to any LAN outlet on your office wall. To connect to z/OS TCP/IP, you must have a fixed IP address that is valid on your physical LAN segment. The underlying PC Linux system is a DHCP client, and these differences can be confusing.

The x3270 program is frequently mentioned in this chapter. x3270 is an open-source program that is not distributed by IBM. Various versions can be found on many websites.

The descriptions here involve both the base Linux TCP/IP functions (that use the internal Linux Ethernet support) and z/OS TCP/IP (that use OSA interfaces). They are two separate IP networks, and you must *always* be aware of which one is being described. Without this awareness, the descriptions can be confusing.

Attention: VPN usage on a channel that is used for emulated OSA operation, especially if enabled or disabled when zPDT is active, can cause OSA connections to fail. You should either have a stable VPN (already running when zPDT is started) or avoid using emulated OSA on channels involving VPN.

[2] In strict Linux terminology, we do not have a tunnel interface. We use a *tap* interface rather than a *tun* interface. We use the word *tunnel* in a more generic sense. The tunnel provides a pseudo-LAN adapter.

7.1.1 ISV zPDT 3270 interfaces

There are two ISV zPDT 3270 interfaces for z/OS:[3]

- The aws3274 device manager accepts TN3270e connections[4] from the Linux Internet Protocol network.[5] The Linux TCP/IP port number for this connection is specified in the **`3270port`** parameter in the device map (devmap). z/OS sees these 3270 sessions as local, channel-attached, and non-SNA DFT terminals. Such terminals are suitable for z/OS operator consoles and IBM Virtual Telecommunications Access Method (VTAM) usage. z/OS TCP/IP is not involved and does not need to be running. No emulated OSA devices are needed.
- z/OS TCP/IP provides TN3270e connections through OSA connections. Terminals that are connected this way are not usable as z/OS operator consoles. TN3270e connections through z/OS TCP/IP are routed through VTAM and can be used as Time Sharing Option (TSO) terminals, IBM CICS terminals, and so forth. The OSA-Express functions are emulated by the awsosa device manager.[6]

Specific limitations

ISV z/PDT, when used with the current z/OS system, has specific limitations:

- Generic VLAN Registration Protocol (GVRP) is not supported.
- At the time of writing, Linux *bonding* of several LAN adapters to create a single virtual adapter has not been tested with ISV zPDT.

Attention: The “local” 3270 interface (through the aws3274 device manager and the 3270port number in the devmap) is generally the easiest to use. If your LAN connection can access the base Linux system running ISV zPDT, then a 3270 emulator connection to the specified port number should work. For new zPDT owners (or for ones with no background in LAN setups), leave any exploration of OSA usage until later.

[3] As a reminder, we ignore non-QDIO operation in this chapter. Non-QDIO usage with Systems Network Architecture (SNA) protocols potentially might be another 3270 interface, although there is no “official” ISV zPDT support for it.
[4] A TN3270 connection (as opposed to a TN3270e connection) is accepted, but extended data stream capabilities are not present and some z/OS functions might not work correctly.
[5] The 3270 sessions might be on the base Linux PC or on other PCs that are connected to the base Linux PC through the network. The involvement of Linux TCP/IP is not apparent on the base Linux GUI.
[6] We describe awsosa as an OSA-Express6s device manager, but this description is only approximate. This device manager has attributes of the original OSA, OSA-Express, and OSA-Express6s channels that are available on larger IBM zSystems machines.

7.2 OSA paths

The Ethernet adapters that are used by awsosa are identified by the path parameter[7] in the devmap or possibly by an interface name. The ISV zPDT **find_io** command displays the current Linux Ethernet device configuration. A default ISV zPDT path is assigned for most interfaces.

Figure 7-1 shows an example of using a **find_io** command.

```
$ find_io
 FIND_IO for "ibmsys1@linux-8jfl"

         Interface          Current          MAC                IPv4             IPv6
 Path    Name               State            Address            Address          Address
------   ----------------   ---------------- -----------------  ----------------  ---------
  F0     eth0               UP, NOT-RUNNING  50:7b:9d:ac:73:45  *                *
  F8     wlan0              UP, RUNNING      e4:b3:18:c9:11:a2  192.168.1.108    xxxxxxx
  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  *                *
  ...

         Interface                          Current Settings
 Path    Name               RxChkSum      TSO     GSO     GRO     LRO    RX VLAN      MTU**
------   ----------------   ---------------- -----------------  ----------------  ---------
  F0     eth0                 On*         On*     On*     On*     Off      On*        1500
  F8     wlan0                Off         Off     On*     On*     Off      Off        1500
.

 *  Enabling these functions might lead to poor ISV zPDT Performance,
    please refer to your ISV zPDT documentation for details.

 ** To Enable Jumbo Frame Support, this MTU value and the MTU value for the
    Host Operating System must be set to > 1500.
 End of FIND_IO
```

Figure 7-1 Output from a find_io command

A default path (first column) might not be shown for all the listed interfaces. The rules are as follows:

- Every OSA definition in a devmap must have a unique path that is specified.
- The path names that are displayed by **find_io** are the default values, and they are used if an interface parameter is not specified in the OSA definition in the devmap.
- The default paths A0 - A7 are used only for tap (tunnel) interfaces. Path A0 corresponds to tap0, A1 corresponds to tap1, and so forth.
- The default paths F0 - FF are assigned for other interfaces. If more than 16 other interfaces exist, they must be specified with interface names in the devmap.
- The Ethernet interface names are listed in alphabetical order, with exceptions. Nomenclature such as `ethx`, `em[1,2,3,4]`, `emp`*`nsnn`*, and `p<slot>p<port>` cause the system board interfaces to be listed first.
- Wireless interfaces with names like `wlan[0,1,2,...]` are assigned after the other interfaces. If there are fewer than eight other interfaces, the wireless path assignments begin with F8.

[7] In this context, we can use the term CHPID instead of path to relate more closely to large IBM zSystems terminology.

- Interface names br[x] and virbr[x] might be listed but not assigned default paths because these interfaces generally are not used for OSA Express emulation.[8]
- Interfaces for pan[x] might be listed and assigned default path names but have not been investigated or tested for ISV zPDT usage.
- Linux alias addresses are not listed.

The other **find_io** details (status, MAC address, and IP addresses) are informational. The IP addresses are the ones that are used by the base Linux machine. z/OS (or another IBM zSystems operating system) running within an ISV zPDT instance uses different IP addresses for these interfaces.

Important: The "path" information (such as F0) that is displayed by **find_io** or specified in a devmap is based on Linux information. In some situations, this information (and the associated "path" name for **find_io** and devmaps) can change, which creates an awkward situation for ISV zPDT. The more common problems relate to VPN connections that occur while ISV zPDT is active or complex container situations. The ISV zPDT owner must control or at least understand such environmental changes before starting ISV zPDT. In general, do not alter your VPN state while using zPDT OSA connections.

Also, the "path" identifier represents a hexadecimal number. For example, path=D7 is a valid hexadecimal representation, but path=G1 is not valid. (D7 and G1 are not used in normal zPDT setups.)

Additional information lists offload settings for various offload functions (**tso**, **gso**, **gro**, **lro**, and so forth) that can be controlled with the **ethtool** command. The maximum transmission unit (MTU) that is listed is for Linux, not the hosted IBM zSystems operating system.

The devmap **name** statements for OSA devices might look like one of the following ones:

```
name awsosa 0013 --path=A0 --pathtype=OSD --tunnel_intf=y
name awsosa 0023 --path=F0 --pathtype=OSD
name awsosa 0033 --path=FF --pathtype=OSD --interface=wlan0
name awsosa 0043 --path=E6 --pathtype=OSD --interface=eth0
```

Consider the following details for these examples:

- The first example is a normal tunnel definition and uses interface tap0, which corresponds to default path A0.
- The second example uses path F0, which corresponds to whatever interface **find_io** shows is associated with default path F0.
- The third example uses the **--interface** parameter to associate a Linux interface (wlan0) with an arbitrary 2-byte hexadecimal path name (FF). (This arbitrary path name must not conflict with other path names in the devmap.)
- The fourth example illustrates that any path can be assigned to an interface.
- Only one path can be assigned to an interface.
- The MAC addresses for tap devices are arbitrary and not usually meaningful.

The IP address that is used by z/OS for OSA Express operation is set by the **TCPIP PROFILE** parameter for z/OS TCP/IP or the equivalent when using a different IBM zSystems operating system. These addresses are not shown by **find_io** because they are not known to the base Linux.

[8] We have seen cases where bridge devices seem to create routing problems.

OSA operation through a tunnel can have more parameters in the awsosa name statement:

```
name awsosa 50 --path=A0 --pathtype=OSD --tunnel_intf=y --tunnel_ip=10.1.1.1
    --tunnel_mask=255.255.255.0
```

The **`--tunnel_ip`** and **`--tunnel_mask`** defaults are shown in Table 7-1.

Table 7-1 The --tunnel_ip and --tunnel_mask defaults

CHPID	Linux name	Default IP address	Default IP mask
A0	tap0	`10.1.1.1`	`255.255.255.0`
A1	tap1	`10.1.2.1`	`255.255.255.0`
A2	tap2	`10.1.3.1`	`255.255.255.0`
A3	tap3	`10.1.4.1`	`255.255.255.0`

And so forth through AF and tap15.[9]

In general, the multiple tunnel interfaces are intended for use with multiple concurrent ISV zPDT instances. The same tunnel interface cannot be shared by multiple concurrent ISV zPDT instances. The default IP address for the tap devices (such as `10.1.1.1`) is the IP address at the Linux end of the tunnel. The IP address at the z/OS end can be anything, but generally should be on the same subnet. We typically use addresses such as `10.1.1.1` (Linux end) with `10.1.1.2` (z/OS end).

7.3 Scenarios

A LAN setup can become complex because many variations are possible. We selected five basic scenarios as possible starting points. As a best practice, first use scenario 1, which has no IBM zSystems TCP/IP functions.[10] The second scenario is a simple migration from the first one. The third, fourth, and fifth scenarios provide different ways to connect ISV zPDT functions to an external network. The key difference between these last three scenarios is whether you have a fixed (external) IP address that can be used with your z/OS (or z/VM or Linux for IBM zSystems).

This chapter does not address more complex LAN use, such as might be used for multiple guests under z/VM. Again, as a best practice, start with the basic scenarios that are described in this chapter. After working through these scenarios, you should be familiar with the elements of LAN use that are unique to ISV zPDT.

Important: LAN setup is not part of the ISV zPDT product. The examples in this chapter might help you decide how to configure your TCP/IP setup, but *you* must provide the networking skills to verify and implement your own design.

If relevant, you *must* understand what is implied by an "assigned, fixed IP address": Unfortunately, this terminology can be used for IP addresses on a local "private" internet or on the worldwide internet; these two different environments have different rules and limitations about IP address usage. This situation has implications about external routing, which physical LAN segment you use, domain and hostnames, and other factors. This situation is not simple and typically is not something a basic user can create.

[9] The path names are expressed in hex, and the Linux interface names have decimal suffixes.
[10] If you follow the basic installation instructions in this publication, you will be close to this environment.

7.4 Basic QDIO setup for z/OS

Consider the following information for QDIO operation:

- Three OSA device addresses are needed for a z/OS TCP/IP connection. The first should be at an even-number address (device number).[11]
- The z/OS devices must be defined as OSA devices in the z/OS input/output definition file (IODF).

  Recent z/OS Application Development Controlled Distribution (ADCD) systems include OSA devices starting at device number 400. When using the QDIO interface to the emulated OSA-Express function, the key parameters might look like the example in 7.14, “A QDIO example” on page 174.
- A TRLE definition is needed in VTAMLST, and pointed to by ATCCONxx in VTAMLST.

  A VTAM major node known as a TRL is required in VTAMLST for QDIO operation. This VTAM node must be active before TCP/IP can use it. The VTAMLST ATCCONxx member must point to the TRL entry in VTAMLST. The link between the TRLE (which specifies the OSA hardware addresses) and the TCP/IP PROFILE parameters in z/OS is an arbitrary name. In the TRLE, it is the PART name; in the z/OS TCP/IP PROFILE entry, the name appears in the DEVICE name (second field), the LINK parameter (fourth field), and the START statement. The name is arbitrary, but it must be the same in all four places. The example in 7.14, “A QDIO example” on page 174 uses PORTA and PORTB as these arbitrary names.
- The z/OS TCP/IP PROFILE uses an IPAQENET device type.

7.5 Five scenarios

Five scenarios are described in this chapter. We use z/OS as the target operating system in these descriptions, but z/VM or Linux for IBM zSystems can be used with appropriate adjustments. QDIO usage in z/OS requires parameters in VTAMLST, which are included in the setup examples.

The five scenarios are numbered as follows:

1. No TCP/IP function is used in z/OS. Only emulated local 3270 connections are used between the base Linux and z/OS. The base Linux can be connected to a larger LAN, which is transparent to z/OS. Emulated 3270 sessions from the base Linux (or from the LAN through the base Linux) can connect to z/OS, where they appear as local, channel-attached 3270 sessions.
2. z/OS TCP/IP is connected to the base Linux through a tunnel function.[12] All z/OS TCP/IP activity is directed to the tunnel, which allows TCP/IP connections between the base Linux and z/OS, which might be used for FTP, Telnet (to UNIX System Services), TN3270e connections directly to z/OS TCP/IP, and so forth. The base Linux can be connected to an external LAN, but this setup is transparent to z/OS and external LAN connections cannot be made to z/OS. With this setup, a TSO user can log in through the aws3274 device manager or the z/OS TCP/IP interface through OSA.

[11] The “even-numbered address” seems to be required by z/OS. It might not be required by z/VM.

[12] The correct terminology is “connect through a Linux tap interface.” However, we use the term *tunnel* in a generic sense to describe this connection.

3. The same basic setup as scenario 2, but with more customization to enable a simple Network Address Translation (NAT) function in the base Linux, which permits z/OS TCP/IP connections to an external LAN, but not from the external LAN to z/OS. (Only outgoing TCP/IP sessions can be initiated.) With more NAT/**`iptables`** customization, incoming TCP/IP connections from the external LAN to z/OS can be handled. This additional customization might involve non-standard port numbers for either Linux or z/OS.
4. Instead of the NAT functions that are used in option 3, an additional OSA interface is used by z/OS to connect to the LAN. A fixed IP address is needed for z/OS. TCP/IP communication between z/OS and the base Linux is through the tunnel.
5. A different NAT function is used that allows incoming and outgoing connections to z/OS. In this scenario, only the tunnel OSA is used by z/OS and both tunnel and external LAN traffic flow through it. The z/OS setup is the same as for scenario three, but the base Linux setup is different. A fixed IP address is needed for z/OS.

In these scenarios, the names that are assigned to the OSA interfaces for z/OS are eth1 and eth2 (if needed). These examples use eth1 for the tunnel connection to the base Linux.

OSA definitions for ISV zPDT require a path parameter. The path for the tunnel is assumed to be A0 and the path for the external LAN is assumed to be F0. Verify these paths with the ISV zPDT **`find_io`** command. This command might not display information for tap devices until after ISV zPDT is started at least once with a tunnel definition that is included in the devmap.

7.5.1 Scenario 1

Scenario 1 is illustrated in Figure 7-2. (The two “boxes” that are shown represent different parts in the same base Linux system. The ISV zPDT system, running in the same base Linux, is shown as a separate box to help make the text descriptions clearer.)

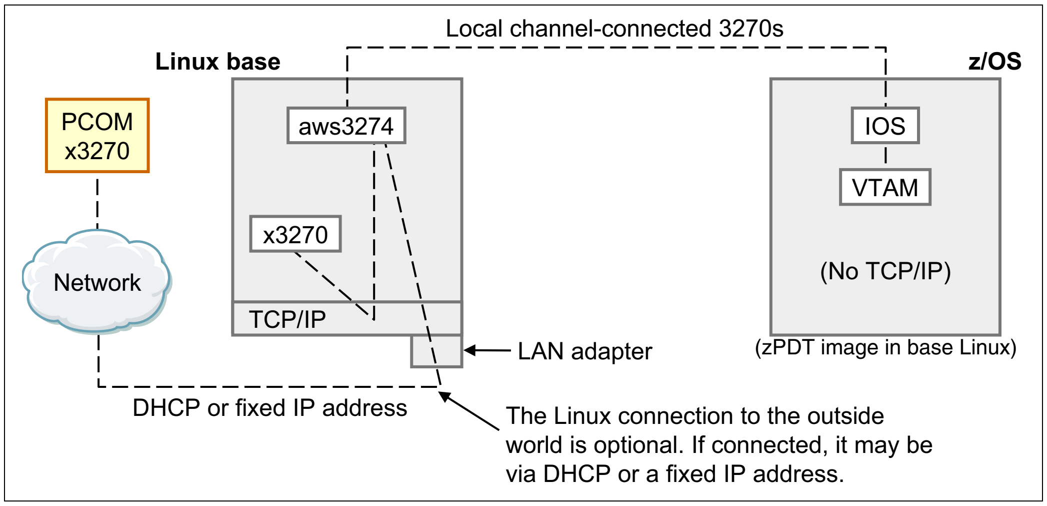

Local channel-connected 3270s
Linux base
z/OS
PCOM x3270
aws3274
IOS
VTAM
x3270
Network
(No TCP/IP)
TCP/IP
LAN adapter
(zPDT image in base Linux)
DHCP or fixed IP address
The Linux connection to the outside world is optional. If connected, it may be via DHCP or a fixed IP address.

Figure 7-2 Scenario 1 connectivity

With this option, no z/OS TCP/IP setup is required and z/OS TCP/IP does not need to be active. (The ADCD z/OS system starts TCP/IP by default. You can remove the associated **`start`** statement in the parmlib member if you do not want the automatic start.) You can use up to 31 TN3270e sessions for connections to TSO or other VTAM functions. (One TN3270e session is normally used for the IBM MVS operator console.)

Various TN3270e emulators can be used, including x3270 (on Linux) and IBM Personal Communications (on Microsoft Windows). These 3270 emulator sessions might be in the base Linux or through a LAN connection to the base Linux. (The LAN connection to an external network is optional.) The only upload/download method between the base Linux and z/OS is by using the IND$FILE functions.[13]

No OSA definitions are needed in the devmap. The relevant devmap definitions are for the 3270 port and for several local 3270 devices, as shown in the following stanza:

```
[system]
....
3270port 3270       #The port number is arbitrary. 3270 is easy to remember.

[manager]
name aws3274 0001
device 0700 3279 3274     #Address 0700 is the MVS console in the ADCD systems.
device 0701 3279 3274     #Other systems might want different addresses.
device 0702 3270 3274
...
```

Based on this example, z/OS connections from the base Linux might start as follows:

```
$ x3270 -port 3270 localhost &    (Assume using an emulated 3270 that is named
“x3270”.)
```

A connection from the external LAN might be started as follows:

```
$ x3270 -port 3270 9.111.222.123 &
```

This command assumes that the external IP address that is assigned to the local Linux is 9.111.222.123. You can find the IP address for your Linux with the **/sbin/ifconfig** or **ip a** command.[14]

Making a TN3270e connection to aws3274 on the base Linux (or any other IP service on the base Linux) from an external LAN might present routing difficulties. The LAN must have route definitions that allow both the external system and the base Linux to find routes to each other. This routing requirement is not unique to ISV zPDT. If you have a firewall running in the base Linux, you might need to create a “hole” in it for the connection to port 3270 (or whatever port that you defined for aws3274 connections). If your firewall is based on **iptables** commands, you can use the following commands:

```
$ su                             (Switch to root.)
# iptables -I INPUT -p tcp --dport 3270 -j ACCEPT
# exit                           (Leave root.)
or
# firewall-cmd --add-port=3270/tcp --zone=public
```

These commands are entered through a Linux terminal window. In general, details about managing your Linux firewall and your external routing controls are beyond the scope of this publication.

[13] IND$FILE functions are often known as file transfer functions in the 3270 emulators.

[14] However, if your Linux is connected to a local router, the DHCP address might be valid only in the local network that is created by that router. IP addresses in the 10.xxx.xxx.xxx and 192.168.xxx.xxx range are usually in this category.

7.5.2 Scenario 2

Scenario 2 builds on scenario 1, and adds a direct TCP/IP connection between z/OS and the standard part of the base Linux, as shown in Figure 7-3.

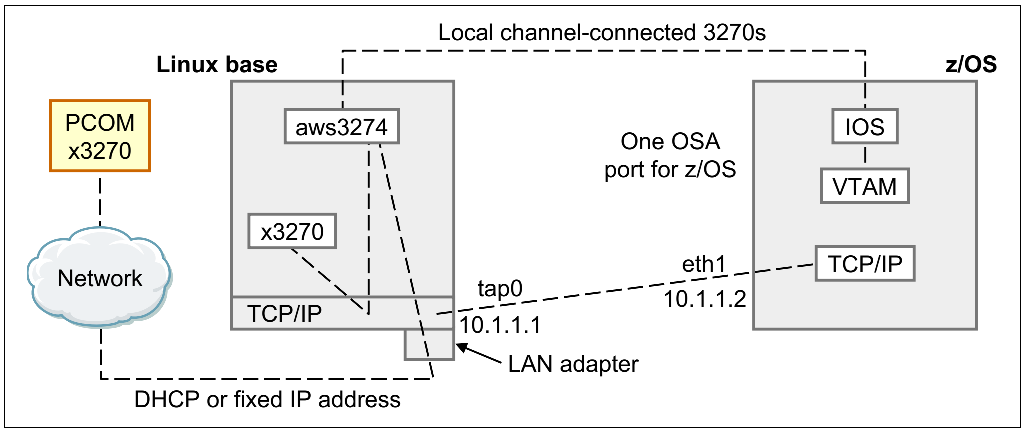


Figure 7-3 Scenario 2 connectivity

This TCP/IP connection is through a “tunnel” interface between z/OS and the base Linux. The physical LAN adapter is not involved. The `10.x.x.x` IP addresses that are shown are arbitrary, but as a best practice, use non-routable addresses on an isolated subnet. The tap interface (and the associated IP address) is created automatically when ISV zPDT is first started (assuming that the correct OSA definitions are in the devmap). No additional Linux setup is needed. z/OS TCP/IP must include an OSA definition for its interface.

ADCD z/OS systems include OSA devices starting at device number 400. When using the QDIO interface to the emulated OSA-Express6s function, the key parameters might look like the following example:

- Devmap

  The 3270 port and aws3274 device manager definitions that are used in the previous example should be included here.

```
[manager]
name awsosa 22 --path=A0 --pathtype=OSD --tunnel_intf=y
device 400 osa osa
device 401 osa osa
device 402 osa osa
```

- z/OS VTAMLST TRL definition[15]

```
OSATRL1  VBUILD TYPE=TRL
OSATRL1E TRLE   LNCTL=MPC,READ=(400),WRITE=(401),DATAPATH=(402),       X
                PORTNAME=PORTA,MPCLEVEL=QDIO
```

- z/OS TCP/IP profile

```
...
DEVICE PORTA MPCIPA
LINK ETH1 IPAQENET PORTA
HOME 10.1.1.2 ETH1
... <== (Many port addresses usually are defined here.)
BEGINRoutes
;     Destination  Subnet Mask    FirstHop    Link  Size
```

[15] The VTAMLST ATCCONxx member must point to this TRL member.

```
   ROUTE 10.0.0.0    255.0.0.0          =        ETH1 MTU 1492
   ROUTE DEFAULT                   10.1.1.1      ETH1 MTU 1492
   ENDRoutes
   ...
   START PORTA
```

The external LAN that is connected to Linux and the “tunnel LAN” between Linux and z/OS are separate in this example, and there is no communication between them. There is no connection from z/OS to the outside world, but all normal TCP/IP functions between the base Linux and z/OS can be used. Examples (from the Linux side) include:

```
$ x3270 -port 3270 localhost &       (Connect through the “local 3270” channel.)
$ x3270 10.1.1.2 &                   (Connect through z/OS TCP/IP.)
$ ftp 10.1.1.2                       (Connect to z/OS FTP.)
$ telnet 10.1.1.2 1023[16]           (Connect to z/OS UNIX System Services.)
```

From the z/OS TSO side, you might use a command such as this one to connect to FTP on the base Linux (assuming that FTP is installed and available on the base Linux):

```
ftp 10.1.1.1                         (entered in ISPF option 6, for example)
```

Tip: We use the **DEVICE** statements in our TCP/IP PROFILE examples. You can use the **INTERFACE** form instead. For example:

```
INTERFACE ETH11
  DEFINE IPAQENET
  IPADDR 10.1.1.2
  PORTNAME PORTA
```

Tunnel IP addresses

The IP addresses that are used for the tunnel (`10.1.1.1` and `10.1.1.2` in the examples) are not related to any other IP addresses that you might use. They are not related to any external IP addresses. They should not be on the same subnet as any external IP addresses that you might use. These tunnel addresses are solely for use between the base Linux and TCP/IP stacks running within the ISV zPDT environment.

The IP address for the base Linux side of the tunnel defaults to `10.1.1.1` (for the first tunnel OSA), but can be changed in the devmap. The address at the other end (z/OS) must be different but should be on the same subnet as determined by the netmask. The `10.x.x.x` addresses (and `192.168.x.x` addresses) are not routable. You should not attempt to make them visible to your external network users.

In our examples, the `192.168.x.x` addresses are assumed to be on the “local side” of a router, which is probably a NAT router. As used in our examples, the `192.168.x.x` addresses are visible and usable by other systems that are connected to the local side of this router.

Important: We again stress that a LAN setup can become complex, especially when “external” connections are involved. This complexity is not related to ISV zPDT. Specific and detailed information about your “external” LAN addressing and routing can be critical. The following details (in this chapter) assume that the reader has some familiarity with LAN operational details.

[16] Some ADCD z/OS releases use port 1023 for a simple Telnet connection to UNIX System Services. You might need to define this function, as described in 12.14, “Telnet” on page 270. SSH can be used instead of Telnet.

7.5.3 Scenario 3

Scenario 3 is depicted in Figure 7-4.

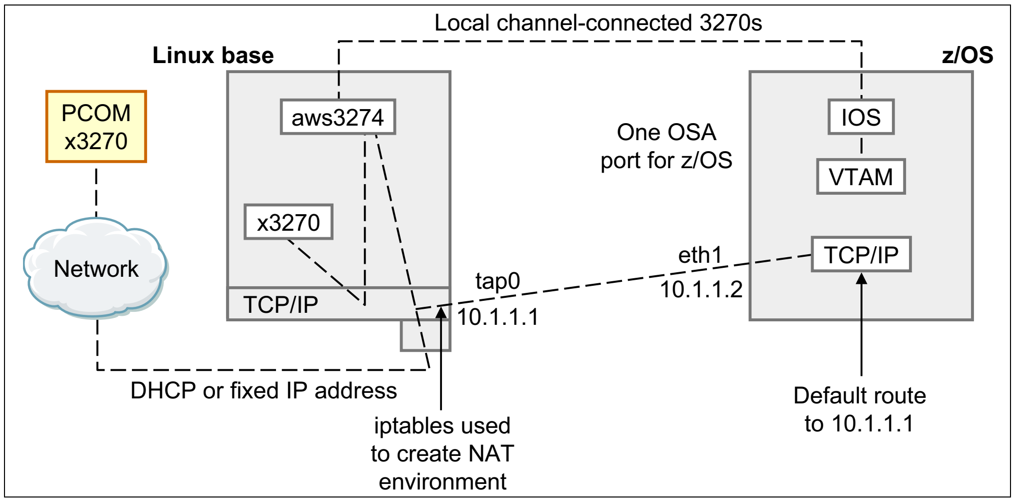


Figure 7-4 Scenario 3 connectivity

You can take the scenario 2 setup and extend it to connect z/OS to the external LAN by using a NAT function in the base Linux, which requires a more complex setup. However, this setup has the major benefit that an external assigned, fixed IP address is not needed for z/OS. With this setup, z/OS has the fixed address `10.1.1.2` (in our examples), but it is not an externally assigned address; rather, it is visible only internally in our local Linux system.

The following sections provide an example to do this extension dynamically (through commands each time that the system is started) by using **`iptables`** commands or **`firewall-cmd`** commands (or another interface that is appropriate for your Linux). These details might vary with different Linux distributions, and you can adapt the principles to your needs.

Working with iptables

The first step is to create a Linux file in the ISV zPDT home directory, as shown here. We arbitrarily named this file `masq`.

```
$ cd ~
$ gedit masq              (The following lines start in column 1 in the file.)
if [[ $EUID -ne 0 ]]; then
  echo 'You must su to root to run this command' 1>&2
  exit 1
fi
echo 'Your firewall must be enabled for this command to be meaningful'
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -F FORWARD
iptables -P FORWARD ACCEPT
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
iptables -I INPUT -p tcp --dport 3270 -j ACCEPT
echo 'Done. Exit from root'
```

The **`iptables`** command is in `/usr/sbin` on some Linux releases. You might need a complete path name for the command. The **`eth0`** operand represents the NIC adapter name on *your* Linux. It might not be eth0. There are many ways to display the NIC name, including the ISV zPDT **`find_io`** command.

Important: The specific **`iptables`** commands that are listed here were suitable for the Linux that we used at the time that we tested them. Linux changes. You might need to explore various approaches to these scenarios that manipulate **`iptables`**.

Use the same devmap and z/OS PROFILE parameters that are shown for scenario 2. Assuming that your base Linux is connected to an external LAN (either with a fixed IP address or a DHCP address), activate your Linux firewall (if not already done) and activate the **`iptable`** changes:

```
$ cd ~
$ chmod 755 masq          (Make it executable.)
$ su                      (Switch to root. You might use sudo.)
# ./masq                  (Run the command script that you created.)
# exit                    (Exit from root.)
```

Working with firewall-cmd

If your base Linux uses the **`firewalld-com`** commands, you can issue something like the following command:

```
# firewall-cmd --zone=external --change-interface=eth0
# firewall=cmd --add-port=3270/tcp --zone=public
```

The *external* zone (of the **`firewalld-cmd`** function) contains a NAT function. The **`eth0`** parameter represents your Linux PC interface on the LAN (you can determine the correct name with **`ifconfig`** or **`ip`** commands). The *public* zone represents your operational zone for Linux. The **`firewall-cmd`** changes that are issued this way are persistent and set during future Linux starts. This method is convenient, but it assumes that you remember what **`firewall-cmd`** environment you set.

Testing

Try the **`ping`** command for external sites. You should be able to ping any internet sites that are known to respond to pings.[17] (You might need the IP address of the target sites.)

You might need to use the **`passive`** option with FTP connections. Complex applications that, once initiated from z/OS, might trigger incoming connections on other ports might not work. Incoming connections to port 3270 on the base Linux are allowed by our script because it provides a *hole* in the standard Linux firewall for using the local 3270 connections. Linux port 3270 (using our examples) goes to the aws3274 device manager, and the connection results in a z/OS 3270 session for the external user.

Incoming connections

The NAT setups have complications for incoming (from the external LAN) connections. In this scenario, external systems see only the IP address of the base Linux, which is typically a DHCP-assigned address. The DHCP address might be unusable for initiating a connection by external users) If an external user attempts to connect to port 23, for example, does the user want Linux port 23 or z/OS port 23? (Assuming that the user can get through the Linux firewall, which is another complication.) Port 23 is the (default) port number for Telnet connections (including TN3270e Telnet).

[17] Our informal tests indicate that many common internet sites no longer respond to pings. You can verify your results by issuing pings from the base Linux system.

One way around this problem is to use a non-standard port number for Telnet on either Linux or z/OS. Another way around the problem is to disallow port 23 connections to either Linux or z/OS. (The issue applies to all port numbers, and we are using port 23 as an example.)

As an example, adding the following line to the `masq` script routes external connections for port 23 to z/OS (assuming z/OS uses the 10.1.1.2 tunnel address):

```
iptables -t nat -A PREROUTING -p tcp -i eth0 --dport 23 -j DNAT --to 10.1.1.2
```

If we add this command to our script, then an external user has two paths for a TN3270e connection (assuming that the Linux IP address is `192.168.1.2`):

```
$ x3270 192.168.1.2 &             #Forwarded to z/OS TCP/IP port 23.
$ x3270 192.168.1.2:3270 &        #Connect to Linux port 3270 (aws3274).
```

After this **`iptables`** command is issued, you no longer have a way to connect to Linux port 23.

Extending this example to other ports, and determining what services might be wanted on both Linux and z/OS, becomes more complex, and depends on the exact base Linux configuration for firewalls and available services.

7.5.4 Scenario 4

This scenario provides a direct connection from z/OS to the external LAN. A NAT function is not used. Only a single physical LAN adapter is needed[18] and can be used by both Linux and z/OS. z/OS must have an external assigned, fixed IP address for this scenario to work. Our example uses address `192.168.1.61`, but it is an example.[19] You *must* have an IP address that is suitable for whatever network you are using.

Figure 7-5 illustrates this configuration. The figure shows two logical connections to the external network, which is accomplished by a single physical cable connection.

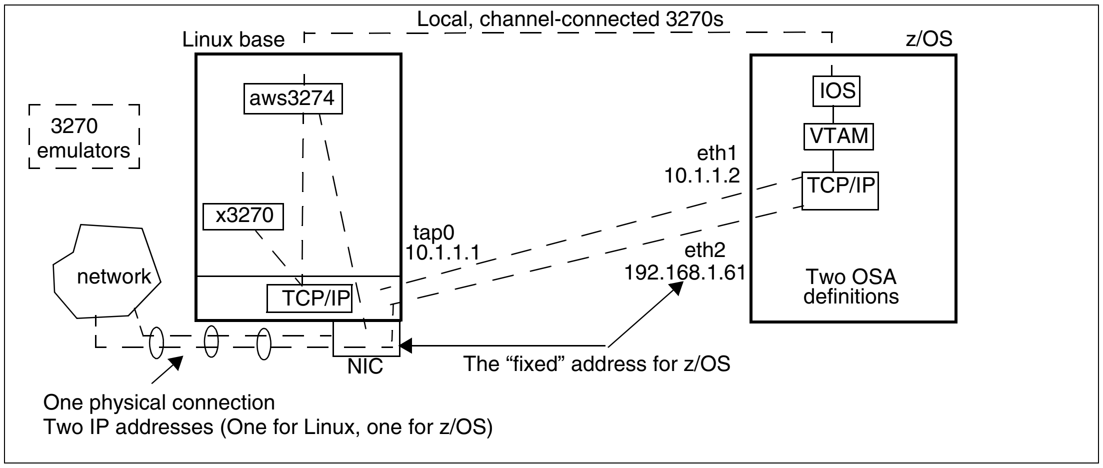


Figure 7-5 Scenario 4 connectivity

With this configuration, the IP functions of z/OS and the base Linux are separate. There are three separate IP connections that are involved: Linux to the external LAN, the 10.x.x.x tunnel, and z/OS to the external network.

[18] You can use a second PC network adapter, but you must avoid external routing loops. If you decide to use a second adapter, first debug your network environment by using a single adapter.

[19] `10.xxx.xxx.xxx` and `192.168.xxx.xxx` addresses are not used with “real” external internet connections. They are intended only for local (non-routed) connections.

The various definition files might contain the following details:

- Devmap

  The 3270 port and aws3274 device manager definitions that are used in the previous examples should be included here.

  ```
  [manager]
  name awsosa 0024 --path=A0 --pathtype=OSD --tunnel_intf=y
  device 400 osa osa
  device 401 osa osa
  device 402 osa osa

  [manager]
  name awsosa 0022 --path=F0 --pathtype=OSD
  device 404 osa osa
  device 405 osa osa
  device 406 osa osa
  ```

- z/OS VTAMLST

  ```
  OSATRL1  VBUILD TYPE=TRL
  OSATRL1E TRLE   LNCTL=MPC,READ=(400),WRITE=(401),DATAPATH=(402),       X
                  PORTNAME=PORTA,MPCLEVEL=QDIO
  OSATRL2E TRLE   LNCTL=MPC,READ=(404),WRITE=(405),DATAPATH=(406),       X
                  PORTNAME=PORTB,MPCLEVEL=QDIO
  ```

- z/OS TCP/IP profile

  ```
  DEVICE PORTA MPCIPA
  LINK ETH1 IPAQENET PORTA
  HOME 10.1.1.2 ETH1

  DEVICE PORTB MPCIPA
  LINK ETH2 IPAQENET PORTB
  HOME 192.168.1.61 ETH2            <==This address is the fixed IP address.

  ...
  BEGINRoutes
  ;     Destination  Subnet Mask    FirstHop    Link  Size
  ROUTE 10.0.0.0    255.0.0.0         =        ETH1  MTU 1492
  ROUTE 192.168.1.0 255.255.255.0     =        ETH2  MTU 1492
  ROUTE DEFAULT                   192.168.1.1   ETH2  MTU DEFAULTSIZE
  ENDRoutes
  ...
  START PORTA
  START PORTB
  ```

With this scenario, connections to and from z/OS and the external network are independent from base Linux connections. However, you must still use the `10.1.1.x` addresses for TCP/IP communication between the base Linux and z/OS, which is why we show two OSA definitions and connections in this example.

Addresses `10.x.x.x` and `192.168.x.x` should not be used for connections to the internet. (`172.16.x.x` - `172.31.x.x` are also in this category, but less commonly used.)

7.5.5 Scenario 5

Figure 7-6 shows connectivity for this scenario.

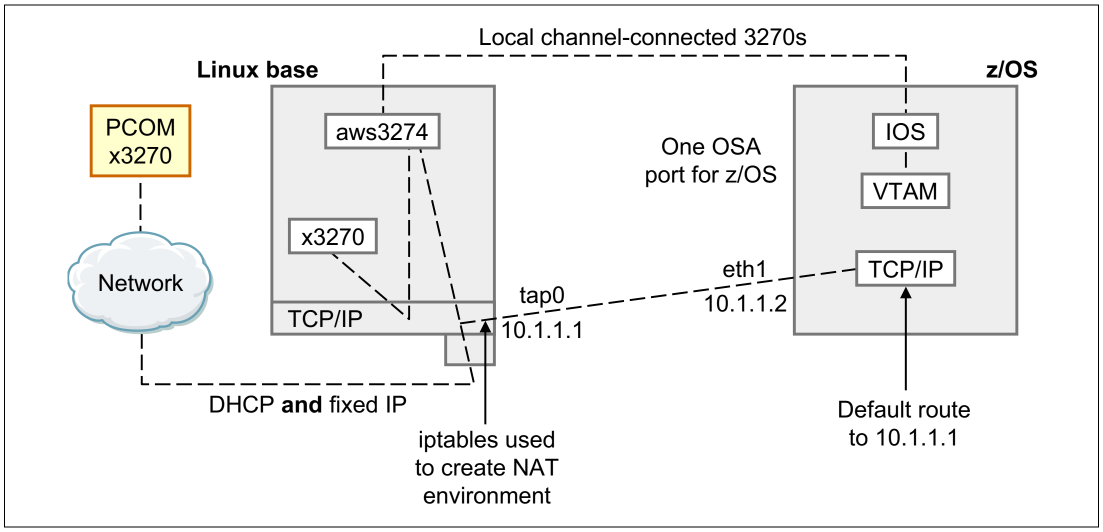


Figure 7-6 Scenario 5 connectivity

You can take the scenario 2 setup and extend it to connect z/OS to the external LAN by using a NAT function in the base Linux in a different way. This method requires an assigned, fixed IP address for z/OS.

A single OSA interface in z/OS handles both tunnel traffic (to the base Linux) and external IP traffic. Incoming connections to z/OS are handled, and outgoing connections.

The following section describes how to do this task dynamically. The example is only for **`iptables`**, so users of **`firewall-cmd`** must devise a similar scheme. This method uses an IP alias address in the base Linux.

The first step is to create a Linux file in the ISV zPDT operational directory.[20] We use a more elaborate script file here to allow it to be expanded in the future. We named this file `nat2`. You must substitute your appropriate IP addresses in the script.

```
$ cd ~
$ touch nat2
$ gedit nat2           (The following lines start in column 1 in the file.21)
if [[ $EUID -ne 0 ]]; then
   echo ‘You must be root to run this command’ 1>&2
   exit 1
fi
echo ‘Your firewall must be enabled for this command to be meaningful’
CHPID_A0_INTERFACE=eth0
CHPID_A0_EXTERNAL_IP=192.168.1.80             (Your assigned IP address)
CHPID_A0_EXTERNAL_BC=192.168.1.255            (Broadcast address for it)
CHPID_A0_EXTERNAL_NM=255.255.255.0            (Netmask for it)
CHPID_A0_VIRTUAL_IP=10.1.1.2

echo 1 > /proc/sys/net/ipv4/ip_forward
```

[20] Or some other convenient directory.

[21] Four lines in this file end with a backslash (\) to indicate that the logical line is continued on the next printed line. You can enter each of these lines as a single long line (without the backslash).

```
echo ‘IP forwarding set’
iptables -t nat -F
echo ‘nat table flushed’

echo ‘External IP address for IBM zSystems is ‘ $CHPID_A0_EXTERNAL_IP
echo ‘Real LAN interface is ‘ $CHPID_A0_INTERFACE
echo ‘Tap (tunnel) address for IBM zSystems is ‘ $CHPID_A0_VIRTUAL_IP
echo ‘External netmask and broadcast address are ‘ $CHPID_A0_EXTERNAL_NM \
$CHPID_A0_EXTERNAL_BC

ifconfig $CHPID_A0_INTERFACE:0 $CHPID_A0_EXTERNAL_IP netmask \
$CHPID_A0_EXTERNAL_NM broadcast $CHPID_A0_EXTERNAL_BC up

iptables -t nat -A POSTROUTING -o $CHPID_A0_INTERFACE -s \ $CHPID_A0_VIRTUAL_IP/32
-j SNAT --to $CHPID_A0_EXTERNAL_IP

iptables -t nat -A PREROUTING  -i $CHPID_A0_INTERFACE -d \
$CHPID_A0_EXTERNAL_IP/32 -j DNAT --to $CHPID_A0_VIRTUAL_IP

echo ‘Done. Exit from root’
```

Use the same devmap and z/OS PROFILE parameters that are shown for scenario 2. Assuming that your base Linux is connected to an external LAN (either with a fixed IP address or a DHCP address), activate your Linux firewall (if not already done) and activate the **iptable** changes:

```
$ cd ~
$ su                      (Switch to root.)
# ./nat2                  (Run the command script that we created.)
# exit                    (Leave root.)
```

You should be able to access external sites from z/OS. External LAN users can connect to your base Linux by using its DHCP address[22] and connect to z/OS by using its fixed address.

7.5.6 Scenario comparison

Table 7-2 summarizes the characteristics of the five scenarios. Note the *from* and *to* words in the descriptions.

Table 7-2 Scenario characteristics

Characteristic Scenario:	**1**	**2**	**3**	**4**	**5**
Number of local 3270 sessions *from* base Linux (including external LAN) *to* z/OS (by using the aws3274 manager).	32	32	32	32	32
Number of OSA definitions in z/OS.	0	1	1	2	1
Number of “command files” that are needed to run in base Linux (**iptables**).	0	0	1	0	1
External LAN connection *from* or *to* z/OS (not counting *local* 3270 sessions).[a]	No	No	Yes[b]	Yes	Yes

[22] You also can have an assigned, fixed address for your base Linux.

Characteristic Scenario:	1	2	3	4	5
TN3270, FTP, or Telnet *from* local Linux *to* z/OS TCP/IP. FTP *from* z/OS *to* local Linux.	No	Yes	Yes	Yes	Yes
TN3270e *from* external LAN *to* z/OS.	No	No	Maybe[c]	Yes	Yes
FTP *from* z/OS *to* external LAN.	No	No	Yes[d]	Yes	Yes
Externally assigned IP address that is needed for z/OS.	No	No[e]	No[e]	Yes	Yes
Telnet connection *to* z/OS UNIX System Services.	No	Yes, only from base Linux	Yes, only from base Linux	Yes	Yes
Browser connection *from* base Linux *to* z/OS.	No	Yes	Yes	Yes	Yes
Browser connection *from* external LAN *to* z/OS.	No	No	Maybe[c]	Yes	Yes
Browser or FTP connection *from* external LAN *to* base Linux.	Yes	Yes	Yes	Yes	Yes

a. The local 3270 sessions are based on LAN connections to Linux (to the aws3270 device manager). The LAN connection is not to z/OS.
b. Only outgoing connections from z/OS to the external LAN can be used unless additional **`iptables`** commands are used. The additional command functions are likely to require the usage of non-standard port numbers.
c. This connection is an incoming one, which is not possible in the most basic **`iptables`** or **`firewall-cmd`** example. Incoming connections can be accepted when an additional **`iptables`** setup is used.
d. This connection is an outgoing connection. A *passive* FTP connection might be needed.
e. The "locally fixed" IP address (`10.1.1.2` in the examples) is not an externally assigned IP address.

The fundamental difference between scenario 3 and scenarios 4 and 5 is that the latter scenarios require an assigned, fixed address for z/OS. Scenario 3 has complications for incoming connections (from the external LAN) to z/OS, but scenarios 4 and 5 do not have this restriction. The primary difference between scenario 4 and scenario 5 is where the external LAN interface appears: With scenario 4, it is defined to z/OS and with scenario 5, it is defined in the base Linux.

7.6 A common error message

When trying to connect a 3270 session to zPDT, you might see the following message:

```
Connection failed:
Transport endpoint is not connected
```

The most common reason for this message is that you (the 3270 user) are not connecting to the port number that is specified by the `3270port` line in the devmap. Check the devmap that you are using and the port number that is used by your 3270 session. A slightly less common reason for the message is that you specified an incorrect IP address for your zPDT system.

7.7 z/OS resolver

Earlier editions of this publication described a detailed method to configure the z/OS resolver on the ADCD z/OS system that was current at the time of writing. Recent ADCD z/OS systems, at the time of writing, have more complex TCP/IP configuration details, and the earlier details are no longer correct. There are several operational areas that are involved:

- Do you want to assign your own z/OS hostname and domain name?
- Does your base Linux system have a correct, fixed IP address as part of an internet connection, or is it a simple Linux system that is connected to a local router that provides DHCP services for it?
- If a DHCP service is provided and used by a local router and your base Linux, does your base Linux properly forward IP connections through the router to an external internet provider?

These options (and others) involve many ways to allow your z/OS resolver to have an acceptable hostname and domain name; resolve an external domain name (such as `www.ibm.com`) to an IP address; and connect to it (such as **`ping WWW.IBM.COM`**). Expanding on these options is outside the scope of this publication. Instead, we provide a simplistic method of using the ADCD z/OS resolver that was current at the time of writing.

We assume the following items:

- You use whatever hostname and domain name is on your ADCD system (such as `S0W1.DAL-EBIS.IHOST.COM`).
- Your base Linux system is connected to a local router that provides DHCP services to your Linux.
- Your base Linux forwards external IP addresses to the DHCP router.
- The router is connected to the external internet through a service provider.
- You allow base Linux IP forwarding and have no firewall stopping it.[23]

With this setup or something similar, complete the following steps:

1. Edit `ADCD.xxxx.TCPPARMS(GBLTDATA)`, uncomment the **`NSINTERADDR`** statement, and give it the value of an internet name server. For example:

   ```
   NSINTERADDR 167.206.10.178            (You might need a different one.)
   ```
2. Stop the z/OS resolver (run **`P RESOLVER`**) and restart it (run **`S RESOLVER,SUB=MSTR`**).
3. Using IBM Interactive System Productivity Facility (ISPF) option 6, run **`ping www.ibm.com`**.

This command should work. The example name server address (`167.206.10.178`) might not be appropriate for you and can be changed. You can expand this task in many ways, but that is beyond the scope of this publication.

7.8 Local router LAN setups

Creating a working LAN environment can be frustrating because many details must work together. Sometimes the problem lies outside your environment, perhaps with external routers that are not configured for the “new” IP addresses that you are placing on the LAN. If your base Linux is in a virtual environment, it adds another complication.

[23] Removing or stopping a Linux firewall appears to be more complex with each new Linux release. This topic is outside the scope of ISV zPDT.

Use a small, inexpensive personal router for the initial ISV zPDT LAN setup in a non-virtual environment, as shown in Figure 7-7.

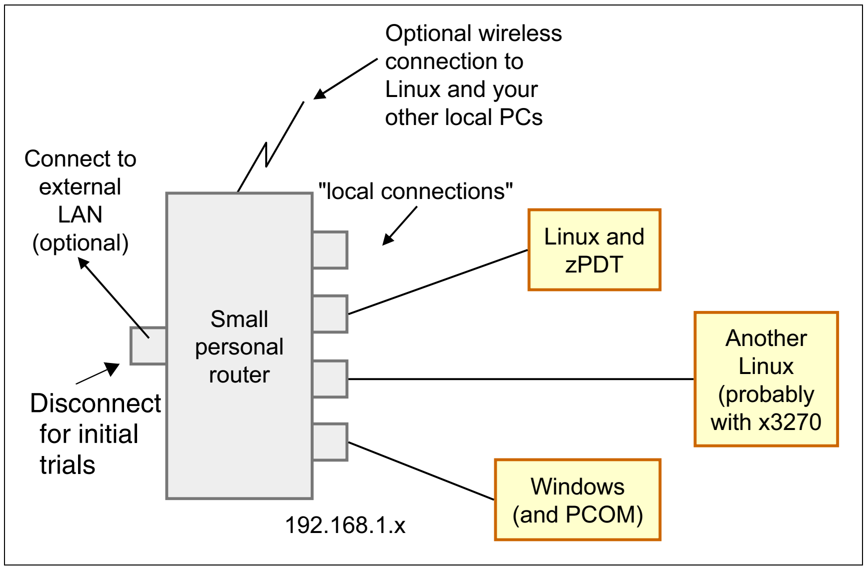

Optional wireless connection to Linux and your other local PCs
Connect to external LAN (optional)
"local connections"
Linux and zPDT
Small personal router
Another Linux (probably with x3270
Disconnect for initial trials
Windows (and PCOM)
192.168.1.x

Figure 7-7 LAN debugging setup

The advantage of using this test setup is that there are no unknown external LAN complications. The “local connections” to the router can be served DHCP addresses by the router, or they can be assigned fixed addresses on the local router subnet. These routers typically use IP addresses such as `192.168.1.x`. Many such routers also provide wireless connections, which you can explore. (Initial use with wired connections simplifies the first setup.)

After an environment like this one is working, you can transfer the operation to a larger network. Although this suggested environment is almost trivial, we often find it useful.

The Linux firewall (if enabled) might affect any connections. For initial debugging (especially in a private environment, as shown here), disable the firewall until you verify that your basic LAN setup is working. Security Enhanced Linux (SEL) might need to be disabled or modified for some functions, such as FTP.

In most cases, the ISV zPDT user has a single network Ethernet interface that is available in their workplace, which is connected to a router that is external to the user. This external network interface typically expects a DHCP client, which presents two problems:

- The IBM zSystems operating system might not operate as a DHCP client.[24] The operating system probably wants a fixed IP address. In general, smaller, network-connected users do not have fixed IP addresses that are valid on the internet.
- The ISV zPDT machine might have multiple LAN adapters, which require multiple network connections.

Most modern personal routers, such as shown in Figure 7-7, contain a NAT function that you can configure. This NAT function allows your machine (base Linux and z/OS) to work with IP addresses that are provided by the NAT router. These addresses typically are in the `192.168.1.xxx` range. The router works with addresses that are provided by your external network (which might be a different set of DHCP addresses).

[24] The Linux for IBM zSystems operating system might provide a DHCP client interface. The “traditional” IBM zSystems operating systems do not function as a DHCP client.

We specified the base router IP address (`192.168.1.1`) as the default gateway address in our TCP/IP definitions (for both Linux and z/OS). For a multiuser system, we connected more PCs to the router (which supplied its own range of DHCP addresses, if requested). The additional PCs can connect to aws3270 (by using the Linux IP address and port 3270) or to OSA (by using the IP address that is assigned, which is specified in the z/OS TCP/IP PROFILE).

Most routers can be configured to pass incoming port connections to specific local IP addresses. This task requires some work with the router software, but allows the handling of incoming connections to z/OS (coming from a DHCP-based external network).

The configuration that is described in 7.5.3, "Scenario 3" on page 160 and 7.7, "z/OS resolver" on page 167 was used with a local router and the **`iptables`** script that is described with the scenario.

7.8.1 Investigating LAN problems

We observed that problems often can be traced to external elements, such as incorrect routing specifications, misconceptions, external LAN performance, or incorrect routing for multiple LAN adapters. These issues are not ISV zPDT issues, and your ISV zPDT provider cannot be expected to debug such problems.

Before asking for help, you should attempt to isolate the problem. Is it really ISV zPDT or is it due to an external element? One way to do this task is to create a small private network, as described here (without the optional external LAN connection). Try this task with a basic Linux and ISV zPDT system, that is, without any virtualization or additional add-on LAN functions. This simple configuration has no external routing and should have no unexpected performance issues.

7.9 Performance problems

At the time of writing, we were aware of two specific problems that might impact OSA performance:

- If frames larger than expected are used, an excessive number of frames might be dropped (causing retransmission). This event might not be noticed unless careful measurements or comparisons are made. We believe that this problem is resolved by including the **`sysctl`** parameter:

  ```
  net.core.rmem_max=1048576
  ```

- Linux attempts to offload various low-level LAN interface functions to the NIC adapter. This action might not be compatible with the awsOSA implementation and can result in drastic performance degradations. More recent Linux distributions *might* not experience these problems.

IBM does not provide performance specifications for ISV zPDT OSA. When working correctly, informal observation indicates that FTP throughput might be in the 5 - 20 MBps[25] range, assuming an unconstrained network in a dedicated environment. If your performance is worse, consider experimenting with the **`ethtool`** commands that are described here.

[25] This range represents z/OS FTP transfer rates to a base Linux system (on a separate PC) working over a local, unconstrained network.

One or more of the following commands, intended to disable the Linux offloading of IP functions, might improve the situation:[26]

`# ethtool -K eth0 rx off`	Disable RX checksumming offload.
`# ethtool -K eth0 tso off`	Disable TCP segmentation offload.
`# ethtool -K eth0 gso off`	Disable generic segmentation offload.
`# ethtool -K eth0 gro off`	Disable generic RX offload.
`# ethtool -K eth0 lro off`	Disable large RX offload.
`# ethtool -K eth0 rxvlan off`	If you are using virtual LANs (VLANs).
`# ethtool -k eth0`	Display status of the NIC.
`# ethtool -S eth0`	Display statistics.
`# ethtool -K em1  rx off`	Newer style of NIC naming.
`# ethtool -K enp0s25 rx off`	Newer style of NIC naming.

You might need to experiment with these commands.[27] One user reported the following combination to be the most effective for their system.

```
# ethtool -K eth0 rx off
# ethtool -K eth0 gso off
# ethtool -K eth0 gro off
# ethtool -K eth0 rxvlan off
```

We suspect that effective combinations of these options differ with various Linux levels and with various NIC adapters.

7.9.1 Jumbo frames

Important: Not all Linux distributions support jumbo frames.

A Linux jumbo frame is a LAN frame that is larger than 1500 bytes. Starting with GA8, ISV zPDT supports jumbo frames up to 9000 bytes,[28] potentially providing substantial performance improvements for LAN transfers. To use jumbo frames, you must set two parameters:

- The MTU size in z/OS TCP profile statements (or the equivalent statements for other IBM zSystems operating systems) should be set to the wanted value, such as 8992.
- You also must change the Linux MTU size for the corresponding interface. (The default Linux MTU size is 1500.) The exact method for changing the Linux MTU size varies with different Linux distributions. In all cases, you must determine the interface name (such as eth0, wlan0, or tap0 in the previous **find_io** example).

[26] Some of these ethtool settings might not be effective for your specific adapter, but they do no harm.
[27] We found that Ubuntu accepted only the **gso** and **gro** changes.
[28] For reasons beyond the scope of this chapter, the MTU size for TCP/IP definitions is sometimes set to 8 bytes less than the true MTU size, such as 1492 instead of 1500, and 8992 instead of 9000. The size of 9000 (or 8992) bytes for a jumbo frame is not rigidly defined, but is the most common maximum size that is used.

Your Linux distribution might vary slightly, but the following list illustrates where we made the necessary changes (but these details might change in more recent Linux distribution updates):

– SUSE

  Go to `/etc/sysconfig/network` and look for a file name that reflects the interface that you are using, for example, `ifcfg-eth0`. Edit this file and look for a line like the following one:

  ```
  MTU=''
  ```

  Change it to `MTU='8992'`.

– Red Hat

  Go to `/etc/sysconfig/network-scripts` and make a similar change. If an MTU line is not present, add it. The single quotation marks might not be needed.

– Ubuntu

  Use the **`ifconfig xxxx mtu 8992`** command.

A temporary change for any Linux (until Linux restarts) can be made by using the **`ifconfig`** or **`ip`** commands:

```
# ifconfig eth0 mtu 8992              (Use the appropriate interface name.)
# ip link set dev eth0 mtu 8992
```

Jumbo frames are associated with gigabit Ethernet operation, meaning wired LANs, and not wireless WANs. Your LAN connection [29] between your system and whatever systems that you connect to must be capable of handling jumbo frames or the usage of jumbo frames might *reduce* performance. You can test a connection for the usage of jumbo frames by using the following command:

```
$ ping -M do -c 3 -s 8900 my.remote.pal.com   (Use your target URL.)
$ ping -M do -c 3 -s 8900 192.168.1.105
```

- The **`-M do`** option prevents **`ping`** from segmenting the packet.
- The **`-c 3`** option causes **`ping`** to be sent three times. (The count number is arbitrary.)
- The -s 8900 option causes **`ping`** to add 8900 bytes of padding to the normal ping packet.[30]

If the **`ping`** fails (typically by hanging until it times out), then your connection does not support jumbo frames. As best as we understand, generally available internet connections do not support jumbo frames.

The examples in this publication use the non-jumbo MTU size of 1492 because the LAN environments capable of handling jumbo frames are not routinely available.

[29] Which includes all the routers and switches between your system and the target system.

[30] There is nothing special about the number 8900. It is less than the 8992 or 9000 that can be used for the MTU and allows extra space for the **`ping`** packet content.

7.10 Local connections to z/OS TCP/IP

If you decide to install the tunnel connection as described earlier, you can connect from the base Linux to z/OS by both Telnet[31] (in line mode) or by a TN3270e client, such as x3270. Using the IP addresses from our examples, the Linux commands are shown here:

```
$ x3270 10.1.1.2 &            (Starts a TN3270 session through z/OS TCP/IP.)
$ telnet 10.1.1.2 1023        (Starts a line-mode Telnet session through z/OS
                              TCP/IP)
```

The 1023 parameter in the **telnet** command specifies the port number that the ADCD UNIX System Services uses for Telnet connections. This port number (1023) is not standard, and probably applies only to some ADCD z/OS systems. We also noticed that **inet** is not automatically started on some ADCD z/OS systems, which can be corrected with the command **s inetd** on the z/OS operator console.[32]

Tip: We recognize that **telnet** is no longer used much, and we mention it as an example. The **putty** command might be another example.

7.11 Choices

Which 3270 connection mode is better? If only 3270 connections are needed (and not more than 32 sessions are needed), then the usage of basic aws3274 connections (shown in Scenario 1) is better. This mode is simpler to set up and does not require OSA or z/OS TCP/IP to be configured or started.

7.12 Automatic configuration

We described the usage of the **iptables** command for implementing NAT functions, and the **ethtool** command for improving LAN performance. To avoid entering these commands every time that Linux restarts, we add them to root's crontab, which results in the following table:

```
@reboot /usr/sbin/iptables -F FORWARD > /dev/null
@reboot /usr/sbin/iptables -P FORWARD ACCEPT > /dev/null
@reboot /usr/sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE > /dev/null
@reboot /usr/sbin/iptables -I INPUT -p tcp --dport 3270 -j ACCEPT > /dev/null
@reboot /usr/sbin/ethtool -K eth0 rx off > /dev/null
@reboot /usr/sbin/ethtool -K eth0 tx off > /dev/null
@reboot /usr/sbin/ethtool -K eth0 gso off > /dev/null
@reboot /usr/sbin/ethtool -K eth0 rxvlan off > /dev/null
@reboot /usr/sbin/ethtool -K eth0 gro off > /dev/null
@reboot /usr/z1090/bin/safenet_daemons_restart reboot > /dev/null
*/11 * * * * /usr/z1090/bin/safenet_daemons_restart > /dev/null
```

[31] We use "telnet" generically, and include SSH in this category.

[32] If appropriate, add this command to the VTAMAPPL script you are using.

The last two lines are created during ISV zPDT installation and must not be altered. The **iptables** and **ethtool** commands were described earlier. The **eth0** operand in many of the lines is the name of our Ethernet port, but the name might be different on your system. The **3270** operand is the port number for aws3274 that is specified in our devmap, but you might use a different port. Our example was used with openSUSE, and you might need adjustments for other Linux distributions.

This example uses **iptables** commands to create a NAT environment. Some more recent Linux bases prefer **firewall-cmd** commands for this purpose, and the above example might not be appropriate.

Important: Again, details are likely to change with newer Linux distributions and updates.

The effect of this table is that after starting Linux (with the firewall enabled), starting ISV zPDT, and performing an initial program load (IPL) of z/OS, you can connect z/OS to external internet sites (due to the NAT operation) and experience reasonable Ethernet transfer rates. (We noticed that the **gro** function is sometimes turned back on automatically; we do not know why this action happens.)

If you are not familiar with crontab customization, do some research before attempting changes to crontab.

Tip: Several earlier editions of this publication described the creation of a **systemd** setup to manage the relevant **ethtool** commands. However, apparently some recent Linux releases and updates (at the time of writing) encountered timing issues by using this method, and we removed the **systemd** recommendation from this edition.

A simpler method for using some of the **ethtool** commands might be by using commands in your devmap, as in the following example:

```
command 2 echo “sudo ethtool -K eth1 gso off gro off”
```

However, you should verify that this command has the correct effect on your system.

7.13 Useful z/OS networking commands

The following commands might be useful when working with LAN devices:

- z/OS operator commands:

D U,,,*dddd*,nn	*dddd* is the address, and nn is the number to display.
D M=DEV(dddd)	Provides the path status.
D M=CHP	Displays all CHPIDs that are defined to z/OS.
D IOS,MIH	Displays the Missing Interrupt Handler (MIH) values.
SETIOS MIH,DEV=E201,TIME=00:30	Example of setting MIH.

- TSO commands:

NETSTAT DEV	Displays all devices and links.
NETSTAT HOME	Displays the home address.
NETSTAT GATE	Displays gateway addresses.
NETSTAT CONN	Displays the connection status.
TRACERTE ipaddress and **PING ipaddress**	To receive more data, issue **ALLOC DD(SYSTCPT) DA(*)** before **TRACERTE** or **PING**.

- VTAM commands:

V NET,ACT,ID=LCL701	Varies the local 3270 as active to VTAM.
D NET,MAJNODES	Displays major nodes.
D NET,ID=xxxxxx,E	Displays information about a specific node.
D NET,TRL	Lists the TRLEs.
D NET,TRL,TRLE=OSATRL1E	Displays data about a specific TRLE.
V NET,ID=OSATRL,ACT	Activates a major node.
V NET,ID=OSATRL,INACT and **V NET,ID=ISTTRL,ACT,UPDATE=ALL**	Removes inactive TRLEs from the TRL list.

The name LCL701 in the sample **V NET** command is the VTAM name of the terminal. This name is *not* related to the LUname that is specified in the ISV zPDT devmap. A 3270 session has both an aws3274 LUname (specified in the ISV zPDT devmap) and a VTAM name (specified in VTAMLST). Also, MVS operator consoles are not specified in VTAM and have no VTAM name. This terminology is unfortunate because the aws3274 LUname, which is used to link a TN3270e session to an aws3274 definition, is not necessarily the same LUname that is associated with a VTAM operation.

Also, ISV zPDT does not support the virtual MAC (VMAC) function from z/OS. The only virtual MAC that is supported is generated on z/VM with the layer-2 vswitch.

7.14 A QDIO example

We used the **find_io** command to determine that our Ethernet adapter was eth0, and that it was assigned as CHPID F0. The tunnel interface is usually CHPID A0. We decided to use the QDIO mode for both OSA interfaces. We used the following devmap:

```
[system]
memory 8600m
3270port 3270
processors 2

[manager]
name aws3274 0002
device 0700 3279 3274
device 0701 3279 3274
device 0702 3279 3274
device 0703 3279 3274
```

```
[manager]
name awsckd 0001
device 0A80 3390 3990 /z/ZCRES1
device 0A81 3390 3990 /z/ZCRES2
etc

[manager]
name awsosa 0013 --path=A0 --pathtype=OSD --tunnel_intf=y
device 400 osa osa
device 401 osa osa
device 402 osa osa

[manager]
name awsosa 0003 --path=F0 --pathtype=OSD
device 404 osa osa
device 405 osa osa
device 406 osa osa
```

The following lines are in VTAMLST member OSATRL1, and pointed to by the parameters in ATTCONxx:

```
OSATRE1  VBUILD TYPE=TRL
OSATRL1E TRLE  LNCTL=MPC,READ=(0400),WRITE=(0401),DATAPATH=(0402),    X
               PORTNAME=PORTA,MPCLEVEL=QDIO
OSATRL2E TRLE  LNCTL=MPC,READ=(0404),WRITE=(0405),DATAPATH=(0406),    X
               PORTNAME=PORTB,MPCLEVEL=QDIO
```

We used the following TCP/IP profile in z/OS:

```
ARPAGE 5

DATASETPREFIX TCPIP

AUTOLOG 5
    FTPD JOBNAME FTPD1   ; FTP Server
    PORTMAP              ; Portmap Server
ENDAUTOLOG

PORT
     7 UDP MISCSERV            ; Miscellaneous Server
    (there follows a long list of standard service ports)

SACONFIG DISABLED

DEVICE PORTA MPCIPA
LINK ETH1  IPAQENET PORTA
HOME 10.1.1.2 ETH1

DEVICE PORTB MPCIPA
LINK ETH2  IPAQENET PORTB
HOME 192.168.1.82 ETH2

BEGINRoutes
;     Destination  Subnet Mask     First Hop   Link   Size
ROUTE 192.168.1.0 255.255.255.0       =        ETH2 MTU 1492
ROUTE 10.0.0.0    255.0.0.0           =        ETH1 MTU 1492
ROUTE DEFAULT                    192.168.1.1  ETH1 MTU DEFAULTSIZE
```

```
ENDRoutes

ITRACE OFF

IPCONFIG NODATAGRAMFWD

TCPCONFIG RESTRICTLOWPORTS

UDPCONFIG RESTRICTLOWPORTS

START PORTA
START PORTB
```

The ADCD z/OS systems (up to the time of writing) use the **DEVICE**, **LINK**, and **HOME** operands in the TCP/IP PROFILE definition. The same operands are used in this publication to match the ADCD systems. You can use the newer **INTERFACE** parameters if you chose.

VLAN usage

z/PDT OSA emulation supports VLAN usage if the underlying Linux NIC card or NIC driver do not impose their own VLAN control. VLAN works correctly in the systems that IBM uses for tests, but might not work in all systems. Unfortunately, documentation at this level of detail can be difficult to find for some NIC adapters and drivers.

7.15 Base Linux LAN notes

Messages, such as the following example, might be seen in the Linux log (with a **dmesg** command):

```
SFW2-INext-DROP-DEFLT IN=tap0 OUT= MAC= SRC=10.1.1.1 ......
```

These messages are related to the use of multicasting when looking for a DNS name server. The source address that is indicated in the message (`10.1.1.1`) is associated with a tunnel (tap) device in typical ISV zPDT operation, and the address is unlikely to find a DNS server.

These messages do no harm. If your Linux system has no need to find a DNS server (on any LAN interface), you can eliminate the messages by editing `/etc/host.conf`, and changing `multi on` to `multi off`.

7.16 NFS and SMB

There is no explicit ISV zPDT support for NFS or SMB-mounted direct access storage device,[33] although NFS-mounted direct access storage device was used during ISV zPDT development. SMB has not been used, and its usefulness with ISV zPDT is unknown.

In principle, the usage of LAN-accessed files is transparent to ISV zPDT, which sees them as ordinary Linux files. In practice, the user must take care with shared files. The **--shared** option with the awsckd device manager causes it to emulate the operation of *reserve* channel command words (CCWs). This locking protects some z/OS metadata (such as Volume Table of Contents (VTOC) and catalog updates), but does not protect shared data.

[33] SMB is also known as SAMBA.

7.17 Secure (TLS) connections to z/OS

The x3270 terminal emulator can provide a secure (encrypted) connection with z/OS when using the z/OS TCP/IP interface[34] with the appropriate port number. Other 3270 emulators might provide the same service, but the operational details are likely to differ.

ADCD z/OS 2.5

The ADCD z/OS 2.5 system is configured to accept a secure connection from x3270 through z/OS TCP/IP by using port number 2023. This approach does not apply to x3270 sessions through the aws3274 device manager. The following x3270 command might be used:

```
x3270 L:10.1.1.2:2023
x3270 L:Y:10.1.1.2:2023         (Try the "Y" option if a certificate error
                                occurs.)
```

This example uses the `10.1.1.2` IP address that is used for the “tunnel” connections that are referenced throughout this publication. Adjust the address to match whatever IP address is used by z/OS. The 2023 port number is configured in the current ADCD system and represents a secure connection.[35] This usage might require more host certificate setup.

For an example of setting up a secure connection environment for x3270 and z/OS, see Appendix E, “Secure x3270 connection” on page 391.

[34] This material does not apply to 3270 connections through the aws3274 device manager.
[35] This setup for port 2023 might change in future ADCD z/OS systems.

8

ISV IBM Z Program Development Tool licenses

To use Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT), you must have a machine-readable ISV zPDT license and an appropriate IBM zSystems serial number for your ISV zPDT system. If you want to install a z/OS Application Development Controlled Distribution (ADCD) system, you also need a different license to decrypt the distributed initial program load (IPL) volumes. The licenses are provided through a USB token or a remote license server. The serial number is provided in similar ways. For examples of ISV zPDT USB tokens, see Figure 2-1 on page 33.

The installation and usage of these licenses, servers, and serial numbers can appear complex because there are many paths and techniques that might be used. The complexity is lessened if you take a few minutes to understand the basic concepts that are involved.

8.1 Basic concepts

The most basic ISV zPDT configuration (a local ISV zPDT configuration) consists of a single USB token that is connected to the user's personal computer (PC). The token provides the licenses (ISV zPDT and z/OS ADCD installation decryption) and the IBM zSystems system serial number to use, as shown in Figure 8-1.

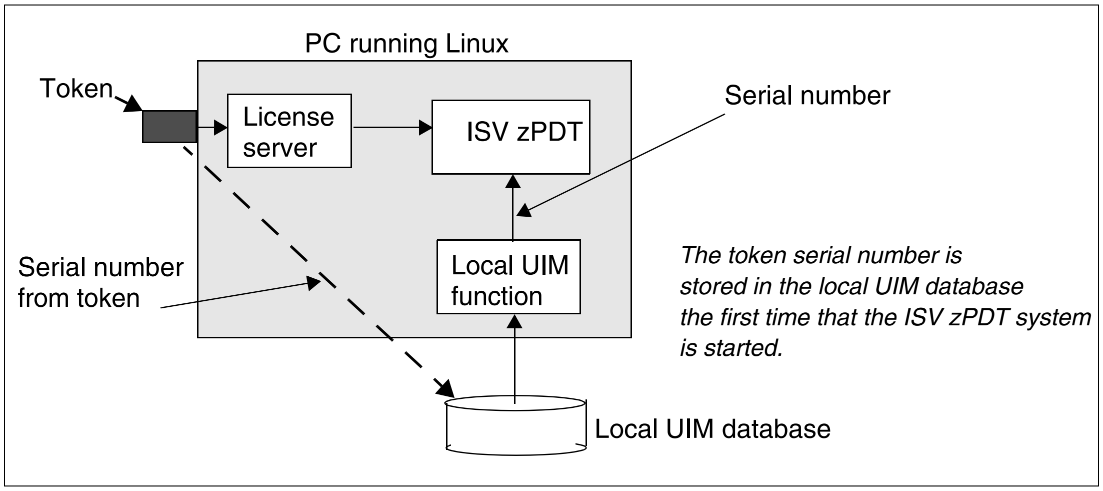


Figure 8-1 Basic ISV zPDT licenses and serial number

IBM zSystems machines have unique serial numbers, allowing software to identify the machine and logical partition (LPAR). The ISV zPDT function that manages serial numbers is known as the Unique Identity Manager (UIM). An IBM zSystems serial number might be important for some users, although most ISV zPDT users accept whatever number ISV zPDT assigns and do not worry about it.[1] In this basic, local configuration, the ISV zPDT license server and the UIM function are both "part of ISV zPDT" and not visible as separate elements. We illustrate them separately to emphasize the functions that they perform.

The UIM data is used only when ISV zPDT is started. The ISV zPDT license is accessed every few minutes while ISV zPDT is operational. Multiple ISV zPDT licenses are required if ISV zPDT uses multiple CPs. A typical token might contain three ISV zPDT licenses and one ADCD license.[2]

The general concept for remote license servers is shown in Figure 8-2 on page 181. The figure also illustrates that multiple instances of ISV zPDT can be used in a Linux client system.

[1] Some operating systems verify that the machine that underwent an IPL has the same serial number as the machine that last used that copy of the operating system and might react differently if there is a mismatch. Some software products are licensed by machine serial number, making it important for users of these products.

[2] The ADCD license is used only to decrypt the z/OS IPL volumes when installing them. It is not needed for routine ISV zPDT or z/OS use. There is seldom a need for more than one ADCD license.

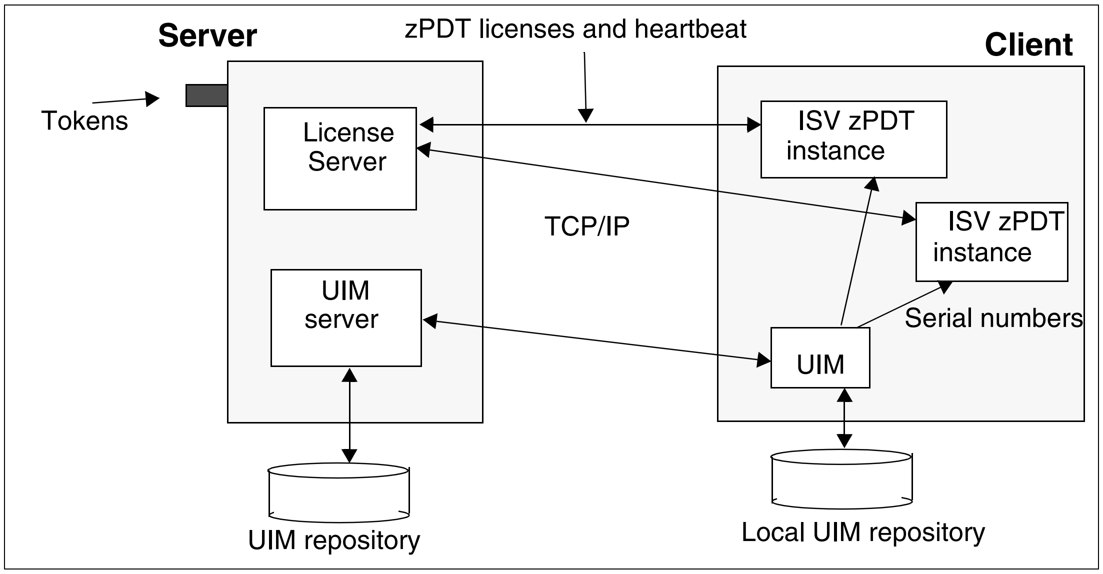


Figure 8-2 General concept of remote licenses and UIM servers

The obvious difference between a local ISV zPDT system and a remote server configuration is that the local machines (the clients) do not have a token that is installed. The license server can provide licenses for multiple clients on multiple PCs, which are limited only by the number of ISV zPDT licenses that the license server has available. The *heartbeat* is a message between ISV zPDT instances and the ISV zPDT license server that, among other things, contains a timestamp.

The network connection between the server and clients is assumed to be reliable. We have seen rare cases where an unusually erratic connection (such as with a high noise level or frequent disconnects) has caused ISV zPDT to hang.

8.1.1 Types of tokens and licenses

There are multiple types of ISV zPDT licenses, tokens, and license servers. There are two general categories of zPDT users:

- Independent software vendors (ISVs), as covered in this publication.
- Commercial customers (IBM Z Development and Test Environment (ZD&T), formerly known as zD&T or RD&T), which is not generally covered in this publication. For more information about this version, see Appendix D, “IBM Z Development and Test Environment notes” on page 387, or check your IBM ZD&T source for more specific documentation.

The remainder of this chapter is devoted to ISV zPDT systems.

The license control mechanisms are from Thales Group (which purchased Gemalto N.V., the previous owner) under the general solution name of SafeNet. There are two SafeNet solution (which we often refer to as products) families, which we refer to as Gen1 and Gen2:[3]

- Gen1 tokens have been used since zPDT first became available, and they include both 1090 tokens for ISV customers and 1091 tokens for ZD&T customers. Gen1 tokens are an older product family.

  Gen1 token type 1090 is used by ISV zPDT. These tokens have 1, 2, or 3 zPDT licenses. The token can be used in a local configuration or in remote license servers. These licenses allow the usage of z/VM and Coupling Facilities (CFs).

- Gen2 licenses are newer and have both a hardware (token) form and a software-only (no token needed) form. At the time of writing, ISV zPDT does not use Gen2 tokens or licenses. If these tokens are required in the future, there will be more documentation about them. (At the time of writing, IBM ZD&T supports the Gen2 software-only license, which is mentioned in Appendix C, "Generation 2 tokens and licenses" on page 379.

  Gen2 tokens and licenses are not covered further in this chapter.

All the 1090 tokens normally contain ADCD licenses. Multiple tokens (of the same type) can be used in both local systems and remote servers to provide more licenses. A single ISV zPDT instance cannot use more than eight ISV zPDT licenses (corresponding to eight IBM zSystems CPs.) For more information, see Chapter 1, "Introduction" on page 1.

8.2 Using a local ISV zPDT system

Using a local ISV zPDT system (that is, one with a token that is connected to a USB port and not connected to a remote ISV zPDT server) depends on the type of token that is involved. Here are the key points:

- If you are using a normal 1090 token, connect it to a USB port. The token must be activated with current licenses. If the token does not contain current license, see 8.11, "Token activation and renewal" on page 196.
- If you are always using the same single token, it is all that is needed. ISV zPDT constructs an IBM zSystems serial number from the token information and stores it in the local UIM database. If you use multiple tokens or different tokens at different times, see 8.3, "UIM usage details" on page 183.

With this basic operation, you can ignore the remainder of this chapter.

[3] These names are informal ones that we use. Some earlier zPDT documentation referred to Gen1 tokens as SHK tokens and Gen2 licenses as LDK licenses.

8.3 UIM usage details

Each ISV zPDT instance is assigned a unique serial number. Every ISV zPDT instance[4] has an LPAR ID that is assigned to it.[5] The combination of serial number and LPAR ID becomes part of the CPUID. The CPUID is the information that is provided by the IBM zSystems system instruction Store CPU ID (STIDP).

The rules for using an ISV zPDT *license* are straight-forward. The rules for ISV zPDT *serial numbers* are more complex. The goal is to always have the same unique serial number for a given ISV zPDT instance.[6] The following general rules are used to determine the IBM zSystems serial number for an ISV zPDT instance. The term *UIM serial number*[7] means a serial number that is generated and assigned by a UIM server. Here are some important details:

- If a single local token is used (and no previous serial was assigned, and the RANDOM option of **`clientconfig`** was not selected):
  - The first ISV zPDT start takes the IBM zSystems serial number from the token. Then, this serial number is written in the local UIM database.
  - Subsequent ISV zPDT starts use the same token:
    - If a different token is used, the **`uimreset -l`** command must be issued first (before ISV zPDT is started). This command erases the existing serial number in the local UIM database, allowing a new token (with a different serial number) to be used.
    - The **`RANDOM`** parameter can be specified by using the **`clientconfig`** command. This command allows any token to be used while retaining a fixed serial number in the local UIM database. (This option must be selected while there is no serial number in the local UIM database. The serial number that is assigned with this option is a random UIM serial number, even though no remote server is involved. It is not related to a token serial number.)
- If a single local token is used and a *UIM serial number* is present in the local UIM database (due to a previous connection to a UIM server or the usage of the **`RANDOM`** option), then the UIM serial number is used and the local token serial number is ignored. (The local token still supplies the ISV zPDT licenses unless a remote license server is configured.)
- If you initially install a simple local system with a single token, the serial number is taken from that token. You can later install an additional token in your local PC to obtain more ISV zPDT licenses, allowing more IBM zSystems CPs to operate. In this case, the serial number from the first token is retained. You do not use the **`RANDOM`** option in this case.

[4] The “instance” terminology is typically used when multiple concurrent ISV zPDT copies (“instances”) are used on a single base Linux.

[5] This ID is not the same as the LPAR name. The LPAR name is the same as the Linux user ID that started the ISV zPDT instance, or a **`system_name`** from the device map (devmap). ISV zPDT instances have some of the characteristics of an LPAR, but full LPAR functions are not provided by ISV zPDT.

[6] Multiple ISV zPDT instances can be used on a single Linux base.

[7] The term *random serial number* is also used for serial numbers that are created by a UIM server. After such a serial number is generated and assigned to a client, it is used consistently. The “random” term applies only to the initial generation of a serial number by a UIM server, which indicates that the serial number is not related to a specific token serial number. You, a client user, cannot create the “random” number.

- If the client is configured for a remote UIM server, the following information applies:
  - If no serial number is known for the client system, the UIM server generates a serial number and sends it to the client UIM database.[8]
  - If the local client UIM database already contains a valid serial number that does not conflict with another client's serial number (as stored in the UIM server database), that serial number is used.

    If the client serial number (in the client UIM database) conflicts with a serial number in the UIM server database, the client operation fails. In this case, the client system can use the `uimreset -l` command to remove the serial number in the local UIM database.
- If the client changes to a local configuration after previously using a remote configuration, the previously assigned serial number (from the remote server and stored in the local UIM database) is used. The local token serial number is ignored.

The user must act to allow a serial number change.[9] A user cannot assign an arbitrary serial number because the serial numbers are generated by UIM or taken from a token.

Important: zPDT does not allow user-assigned IBM zSystems CPU serial numbers.

8.4 General ISV zPDT client and server details

The usage of a remote ISV zPDT license server (usually known as a remote token server) can be convenient. If the remote server is in a more secure area than the client ISV zPDT machines, it helps reduce the chances of the tokens being "borrowed" by someone.

A license or UIM server is accessed (through TCP/IP) by a client PC running ISV zPDT. The ISV zPDT operational license is supplied this way.[10] The client machine does not have a token and does not need a USB port. A client machine must have access to the license server when ISV zPDT is operational on the client. Likewise, the client machine has access to a UIM server that manages consistent serial numbers for the IBM zSystems CPs.

All ISV zPDT systems have remote client functions, but by default it is not configured for remote operation. If the remote client function is configured, ISV zPDT attempts to connect to remote servers to obtain an ISV zPDT license and serial number (if one is not set in the local UIM database).

The owner of the client machine must do some minor configuration work to enable the client to use a remote license server and UIM server. Before enabling client access to a remote server, the server networking environment (IP address, domain name, firewall controls, and appropriate tokens for the server) must be arranged.

The remote license and UIM servers are normally on a single remote system. However, the two servers *can* be on separate machines.[11] A "remote" UIM server or token server can be on the same machine as the client, but this setup is unusual and still considered a remote server in the context that is described here. All the following text assumes that the license server and the UIM server are on the same remote machine.

[8] This serial number is noted as a "random" serial number. In this case, "random" means it is not related to a token serial number. Do not try to generalize the "random" terminology.
[9] Requires the `uimreset` command.
[10] The licenses that are needed to decrypt z/OS IPL volumes are also provided by the server.
[11] This setup is rarely done.

More details include the following ones:

- The TCP/IP port number for a 1090 license server is 9450 and for a UIM server it is 9451. These port numbers are not configurable.[12] If you must change one of these port numbers, contact your ISV zPDT provider for help. Firewalls between the servers and clients must allow the required IP network and port access for both TCP and UDP.
- After an ISV zPDT instance starts (on a client), the license access must be maintained for the life of the ISV zPDT instance. If the access is dropped, the ISV zPDT instance stops. (If the access is recovered, ISV zPDT starts again.)
- The servers must be identified by domain names or by IP addresses. This task is easy if they have direct, fixed IP addresses or domain names. It is not easy if DHCP-assigned addresses, Network Address Translation (NAT) functions, or virtual LAN (VLAN) networks are involved. Skilled network planning is required for any but the simplest environments.
- A client machine can be changed to a stand-alone machine (with a token) by changing a configuration file, and vice versa.
- In normal operation, a client machine always has the same IBM zSystems system serial number. This number, once assigned through a local or remote function, might not be related to any physical token number.
- Any license or UIM configuration changes should be made when ISV zPDT is not operational.

8.5 Client installation and configuration for remote servers

All ISV zPDT client functions (for both licenses and UIM functions) are included and installed by the ISV zPDT installation package. Whether the remote functions are used depends on the configuration options. The basic ISV zPDT client installation process is described in Chapter 5, “ISV IBM Z Program Development Tool installation” on page 125.

8.5.1 Gen1 client configuration

After a normal ISV zPDT installation, the license operation with a remote server is normally configured with the **`clientconfig`** command.[13] The command produces a display like the following one:

```
Gen1 ContactServer........localhost          (Default localhost)
Gen1 BackupServer.........__________________ (Default is blank.)
UIM ContactServer.........__________________ (Default is blank.)
UIM Local Serial Random..._                  (y or blank)
Factory Reset............._                  (Enter “y” to reset file.)
```

[12] This change is one for ISV zPDT GA7.

[13] You must operate as root to use the **`clientconfig`** command or enable the **`clientconfig_authority`** function.

Parameters are changed by typing over them. Configurations for two separate functions (the ISV zPDT license server and UIM server) are specified here. The general rules are the following ones:

- The `Gen1 ContactServer` field value should be `localhost` (to specify that no remote license server is used)[14] or the address (IP address or domain address) of a license server. One of these options must be specified. The default **`localhost`** parameter causes zPDT to operate with a local token. The "Gen1" name is used as an option for future expansion.
- The `Gen1 BackupServer` field is used only if there is a second license server.
- If the `Gen1 ContactServer` field value is not `localhost` (that is, a remote license server is used), the `UIM ContactServer` is assumed to be at the same IP address as the license server. The `UIM ContactServer` value is specified *only* if a UIM server is used *and* is on a different server machine than the license server. In a simple local environment with no UIM server, this line *must* be blank. If used, the `hostname` can be `localhost` (if a UIM *server* is running on the client machine) or the address (IP or domain name) of a remote UIM server. As a practical matter, this line is almost always left blank.
- The **`Random`** option indicates that various tokens can be used without changing the serial number that is stored in UIM.
- If the **`Factory Reset`** option is set to **"**y**"**, all other parameters are ignored, and the configuration file is restored to the original values.

Changes to the configuration file are not dynamic. They take effect only when ISV zPDT is started. Earlier versions of **`clientconfig`** provided a field for a UIM server port number, but this option is no longer relevant.

An alternative method for setting the **`clientconfig`** values for Gen1 and Gen2 licenses is described in 4.2.28, "The clientconfig_cli command" on page 90. This alternative method provides a command-line interface (CLI) that can be used within a Linux script.

The actual server information is in th file `/usr/z1090/bin/sntlconfig.xml`. The general syntax is as follows:

```
<SentinelConfiguration>
        <SentinelKeys>
                <ContactServer>localhost</ContactServer>
                <ServerPort>9540</ServerPort>
                <Protocol>SP_TCP_PROTOCOL</Protocol>
        </SentinelKeys>
        <UniqueIdentificationManager>
                <UIMContactServer></UIMContactServer>
                <UIMServerPort></UIMServerPort>
                <UIMProtocol></UIMProtocol>
                <UIMLocalSerialMethod></UIMLocalSerialMethod>
        </UniqueIdentificationManager>
</SentinelConfiguration>
```

As a best practice, do not attempt to edit this file directly. Directly editing XML files is prone to errors, and debugging can be difficult.

[14] If the `GEN1 ContactServer` field is blank, it internally defaults to `localhost`.

8.5.2 Client UIM configuration

Here are the configuration details:

- For a Gen1 license server, the license server configuration (with the **clientconfig** command) also configures access to the UIM server. By default, the UIM server is assumed to be at the same IP address as the Gen1 server and uses port number 9451. For more information, see 8.5.1, “Gen1 client configuration” on page 185.
- The **clientconfig_cli** line command can be used instead of the interactive **clientconfig** function for these actions.
- The following command clears the serial number in the local UIM database **[-l]** or in both the remote and local UIM database **[-r]**:

  ```
  # uimreset [-l] [-r]                (Must be run by root.)
  ```

- The **uimcheck** command should be used if there is any question about the state of the serial number on an ISV zPDT machine. Any user can issue this command.

  ```
  $ uimcheck                          (May be used by any user ID.)
  ```

8.6 Server installation and configuration

Both the Gen1 license server and UIM server are included in the base ISV zPDT package. The license server runs as a daemon and is automatically started when Linux starts, which is true even for local token use.

8.6.1 UIM server

The UIM server that is used with a Gen1 server is automatically installed when installing ISV zPDT. Once installed, the remote UIM server must initially be started manually; thereafter, it is automatically managed by **cron**. It must *not* run as root. It runs under a normal Linux user ID and places its database in the home directory of that user ID. It also places small log files in the home directory. For this reason, the same Linux user ID (*not* root) should always be used to run the UIM server.

Two commands are associated with running the UIM server:

$ uimserverstart	Start the UIM server.
$ uimserverstop	Stop the UIM server.

Important: Do not use these commands if you are not using a remote UIM server.

The **uimserverstart** command, in addition to starting the server, places entries in the Linux **cron** files such that the UIM server is restarted automatically (after 10 minutes) if it fails. It is also started automatically during a Linux restart. The **uimserverstop** command stops the server and removes these **cron** entries.

No other configuration is needed for the UIM server. You must not edit the UIM database file that is created in a subdirectory of the home directory of the user ID running the UIM server because this action corrupts the file due to checksums that are within it.

If you must change the UIM server port number, consult your ISV zPDT provider.

8.6.2 Gen1 license server

The Gen1 license server is part of the standard ISV zPDT package, and it is installed as though you were installing an ISV zPDT client. It is activated by the actions of the two token "driver" components that are part of ISV zPDT installation.

One (or more) 1090 tokens must be installed in the license server machine before it can be used. The license server configuration file is in the following directory:

```
/opt/safenet-sentinel/common_files/sentinel_key_server/sntlconfigsrvr.xml
```

This file typically does not require any additional configuration. If you must change this file, then you must restart the server:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart
```

Several security functions can be specified in the `sntlconfigsrve.xml` file. However, direct editing of an XML file can be confusing, so we do not recommend this action unless it is necessary.

8.7 General notes

Do not run **`Z1090_token_update`** from a client ISV zPDT machine when using a remote license server. The command cannot affect tokens or licenses in the remote license server, but attempts to access a token in the local PC. You can run the utility in the Gen1 license server to update the tokens in the server.[15]

The client UIM information is held in `/usr/z1090/uim/uimclient.db`. In unusual error situations, you might be advised to delete this file, which causes the UIM function to obtain or create a serial number when ISV zPDT is next started, as described in 8.3, "UIM usage details" on page 183.

The administrator of a license server is responsible for ensuring that the license keys do not expire while in use. The situation in which multiple tokens are installed (in a Gen1 license server) and the licenses in only one token expire can be complex. Clients see license expiration warning messages starting a month before the license expires. However, if multiple tokens are present, it is not predictable which token furnishes the license (or licenses) for an ISV zPDT startup.

The license expiration date that is displayed by the **`token`** command (in a client machine) might not reflect the effective expiration date of all the active tokens in a license server. The **`token`** command (when ISV zPDT is running) produces additional information, for example:

```
$ token
CPU 0, zPDTA (1090) available and working. Serial 6186(0x182A)
  Lic=88570(0x159FA) EXP=4/15/2017 SHK
```

In this example, the ISV zPDT license was obtained from token 0x159FA (decimal 88570) and the CP serial number that was used by ISV zPDT is 0x182A. There is no indication of whether a license server and UIM server are being used. Because the serial number and license number are different, we know that at some point the serial number was obtained from a UIM server or with the **`RANDOM`** option in the local client. (The "SHK" indicates that a Gen1 token is being used.)

[15] Normal guidelines for **`SercureUpdateUtility`** or **`Z1090_token_update`** (or **`Z1091_token_update`**) apply. For example, only one token should be connected to the PC when you use these commands.

Reinstalling your ISV zPDT system

If you decide to reinstall your ISV zPDT system, there might be a problem with serial numbers. If you use the same single local token that was used previously (and do not have the **RANDOM** option set), ISV zPDT obtains the same serial number from it. If you use a remote license server and deleted any previous references there (with an **uimreset -r** command), your new ISV zPDT installation might not have the same serial number as the previous setup. If you do not care about IBM zSystems serial numbers, then this situation is not a problem. If you do care about IBM zSystems serial numbers (due to software contracts or software sensitivity), this situation can be a problem. The only certain way to obtain the same IBM zSystems serial number is to use the same single local token without the **RANDOM** option.

Firewalls

You (or your networking people) must manage any firewalls that are involved with remote servers and ensure that intermediate routers can find your server and your client.[16] If you operate through firewalls, you must ensure that the relevant port numbers can pass through the firewalls. There are many management techniques for firewalls, depending on what product is being used. Many Linux systems respond to **iptables** commands, such as the following ones:

- **# iptables -I INPUT -p tcp - d port 1947 -j ACCEPT**
- **# iptables -I INPUT -p tcp - d port 9450 -j ACCEPT**
- **# iptables -I INPUT -p tcp - d port 9451 -j ACCEPT**

Disk and Linux changes

Changing the Linux disk (hard disk drive (HDD)) might change the identifier that is part of the identification that is used by UIM. You might need to reset the local serial number (**uimreset -l**) or the remote serial number (**uimreset -r**) after changing the HDD.

Upgrading to a new Linux kernel might change the identification that is used by UIM. You might need to reset the local serial number (**uimreset -l**) or the remote serial number (**uimreset -r**). If this action does not solve the problem, delete the UIM database at `/usr/z1090/uim`.

Backup servers

Backup license servers can be specified for Gen1 servers (a token server). Backup servers cannot be specified for UIM servers.

Cloning ISV zPDT

If you clone an ISV zPDT system, you must delete the files in `/usr/z1090/uim` on the new system because the UUID of the new system differs from the one of the old system. ISV zPDT builds new `uim` files when the new system starts.

Removing functions

All Gen1 server functions (and associated UIMs) can be removed by removing ISV zPDT on that server. For example:

```
# z1090-1-7-49.28.x86_64 --removeall       (note two dashes)
```

[16] As we stressed before in this publication, skilled networking help might be required to implement remote access to ISV zPDT or use remote license servers.

License expiration notification

Starting 30 days before an ISV zPDT license expires, ISV zPDT issues a message three times per day in the Linux CLI that was used to start ISV zPDT, that is, the CLI where the **awsstart** command was issued. This Linux window is often not visible to ISV zPDT users or administrators, and a second notification option is available.

If you define environmental variable **ZPDT_EXP_EMAIL** to specify an email address, an ISV zPDT license expiration notification is sent to this address every 8 hours. For example:

```
$ export ZPDT_EXP_EMAIL=bill@my.isp.com
```

The environmental variable should be set from the Linux CLI that starts ISV zPDT and issued before ISV zPDT starts. For this method to work, your Linux **mail** command must be properly configured and operational. This task often is not trivial, and it varies so much that we cannot advise you about how to do it. You can test your **mail** command as follows:

```
$ mail -s “My test message”  bill@my.isp.com      (Use a correct address.)
This is my test message, Line 1
(cntl-D)                               (Use cntl-D to end your message.)
```

If this sequence is successful in sending the test message to your specified address, **mail** is working correctly.

8.8 Scenarios

Common scenarios are as follows:

- License search order
- Local to remote server
- Temporarily switch from server to local
- Remote server to local
- Using ISV zPDT on the license or UIM server (Gen1)
- Switching tokens

License search order

ISV zPDT attempts to obtain a license from a Gen1 server (if one is configured), and then it attempts to obtain a license from a local token. There is a considerable timeout that is involved in trying to access the two servers, and depending on this automatic “fall through”, a search is not reasonable for normal operation. The **--localtoken** option of the **awsstart** command disallows any attempts to use remote license servers and UIM servers.

Local to remote server

Consider ISV zPDT systems A and B (each using a different PC for ISV zPDT with Gen1 tokens). System A has an ISV zPDT token with serial number 12345.

- The system A owner installs token 12345 on their PC and starts ISV zPDT. When this task is done, serial 12345 is recorded in the local system UIM database, assuming that there was no prior conflicting information in the local UIM database and the **RANDOM** option is not used. System A can be used in this configuration indefinitely (until the token license expires), with no reference to remote license or UIM servers.
- The token is taken from system A for some reason, and the system A owner now wants to use remote license and UIM servers. With ISV zPDT not running and working as root, the owner configures a client, as described in 8.5.1, “Gen1 client configuration” on page 185.

- The remote UIM server sees that system A has serial number 12345 recorded in its local UIM database. The server checks whether this serial number is assigned to any other system. If there are no conflicts, the server records serial 12345 in the UIM server database as belonging to system A. Separately, the remote license manager serves an ISV zPDT license, but the serial number of that token (if one is used) is not relevant.

  Thus far, system A has retained a consistent serial number (12345) when switching from a local token to remote license or UIM servers. It will have this serial number every time ISV zPDT[17] is used.
- Someone has given token 12345 to the owner of system B. The owner installs and uses it locally (with no connection to the remote license or UIM servers and without the **RANDOM** option). Now, both A and B have the same ISV zPDT serial number, with system A obtaining its ISV zPDT license from the license server, and system B obtaining its ISV zPDT license from the local token. There is no way to avoid this situation.
- If the system B owner connects to the license or UIM servers, the UIM server sees serial 12345 in B's local UIM database, and terminates the ISV zPDT instance because 12345 already is assigned to system A.
- The problem is that A and B both want to use the same serial number (12345) and the UIM server assigned it to A. There are two ways to resolve this conflict:
  - The system B owner can issue **uimreset -l** to clear the serial number in the local UIM database. Then, the owner can connect to the remote servers and receive a new random serial number.
  - The system A owner can issue **uimreset -r** to clear their serial number from both the local and remote UIM databases. The next time system A's ISV zPDT starts, it requests a new random serial number from the server. Then, system B can use the 12345 token and serial number.

Temporarily switch from server to local

Assume that a notebook ISV zPDT system is normally used with remote license and UIM servers. You want to take the notebook home overnight, and the servers cannot be accessed from home. If a token is available, you can start ISV zPDT with the local option:

```
$ awsstart devmap_name --localtoken
```

In this case, there is no need to use the **clientconfig** command to change the configuration file. The **--localtoken** option overrides the configuration file. The user must have a token to supply a license. In this case, the serial number that is stored in the local UIM database is used and the serial number of the temporary token is ignored. If the remote UIM server is not available (which is likely in this scenario), there will be a short pause before the serial number in the local UIM database is used. (A warning is issued when this action is done.)

Remote server to local

Assume that a system owner is using a remote license server and UIM server. To change to the routine usage of a local token, the owner should use the **clientconfig** command to change the `LicenseContactServer` value to `localhost`. ISV zPDT looks in the local UIM database for a serial number. If one is present, it is used. If the local UIM database does not exist (or if the **uimreset** -l command was used), the serial number of the local token is placed in the local UIM database and then used by ISV zPDT.

[17] To be more precise, we should say "every time this same ISV zPDT instance is used." Multiple ISV zPDT instances (on the same machine) must run under different Linux user IDs. The serial number for each of the instances uses the "LPAR" portion of the serial number to differentiate the instances.

Using ISV zPDT on the license or UIM server (Gen1)

Suppose that you want to run ISV zPDT on the same machine that is running the Gen1 license server and UIM servers. In this case, use the **`clientconfig`** command to specify `LicenseContactServer` as `localhost` and `UIMContactServer` as `localhost`. This action has the following effects:

- The presence of the `UIMContactServer` stanza means that a UIM server must be available on the indicated system (which is `localhost` in this example). Before starting ISV zPDT on this system, the user must issue an **`uimserverstart`** command.

  Think about the Linux user ID that issues the **`uimserverstart`** command. The same user ID always must be used for this command because the UIM server database is created in the home directory of this Linux user ID.

- No special setup is needed for the license server. Any ISV zPDT system (meaning the SafeNet server that is installed with ISV zPDT) can act as a Gen1 license server.

Switching tokens

In this case, token 12345 is used with a newly installed ISV zPDT system. When ISV zPDT first starts, this serial number is written into the local UIM database. If a different token is used on a subsequent start, the ISV zPDT startup fails. An **`uimreset -l`** command is needed to remove serial 12345 from the UIM database, and then a new token may be used.

However, if the serial number in the local UIM database was assigned by a UIM server (or if the **`RANDOM`** parameter was used with the **`clientconfig`** command), then any local token can be used, and the operational serial number will be taken from the local UIM database.

The important point is that ISV zPDT recognizes the difference between a UIM server-assigned serial number (which can be used with any token) and a locally installed serial number (which is taken from a local token). A locally installed serial number must match the token that is used (unless the **`RANDOM`** option is set).

Changing from a single token to multiple tokens

Assume that you start with a single token. Later, you decide that you want to use one of th several tokens that you possess. You are not using a remote license server. To accomplish this task, complete the following steps:

1. Issue an **`uimreset -l`** command. (Requires root authority.)
2. Use the **`clientconfig`** command and set the UIM Local Serial Random value to `Y`. (Requires root authority.)
3. A random serial number is generated the first time that ISV zPDT is started.
4. Now, you should be able to start ISV zPDT with any token or with multiple tokens attached. The serial number that is assigned in step 3 is used regardless of which token you are using.

8.8.1 Security

If the license managers are used only from a single subnet or a VPN, then security is not a major issue. If the license servers are accessed from the general internet, then security can be a significant issue. For example, your license server might provide ISV zPDT licenses to someone external to your enterprise.

Important: License and UIM server port numbers (9450 and 9451 for Gen1) must not be used by any other IP function on your subnet. A port number conflict can cause ISV zPDT failure.

Gen1 server

The SafeNet Gen1 license server can have three lists of IP addresses (or domain names or ranges of IP addresses):

- The *Authorized User List* determines which systems can use a web interface to manage the SafeNet license server. The default list contains only one address, `127.0.0.1`, which is the local host and is always allowed whether it is specified or not.
- The *Allowed Site Address* list determines which clients can obtain ISV zPDT licenses from this server. If the list is empty (the default), then *any* client can obtain a license from this server.
- The *Blocked Site Address* list specifies client addresses that *cannot* obtain a license from this server. If the list is empty (the default), then no client addresses are blocked.

Each list is limited to 32 entries. These lists are in the `sntlconfigsrve.xml` file in `/opt/safenet_sentinel/common_files/sentinel_keys_server/`, and they can be edited there.

Tip: This information also can be displayed or managed by pointing a browser to port 7002 on the machine that is running the SafeNet license server, for example:

```
http://localhost:7002             (if working on the server machine)
```

The license server functions through port 7002 are not part of the IBM zPDT product and the usage is not directly supported by IBM.

If a different machine is used to access the server web interface, then the IP address of that machine must be listed in the Authorized User List. As a best practice, use the browser method, if possible, because directly editing the XML file is prone to introducing syntax errors that might cause the license server to fail. List entries might take any of the following forms:

`127.0.0.1`	A simple IP address
`my.local.domain.com`	A domain name
`10.1.1.2-10.3.255.254`	A range of IP addresses

If you are using the browser interface, be certain to click **Update** on the web page after making the updates to the lists. Then, you must restart the SafeNet server:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart
```

These lists provide one way to secure usage of an ISV zPDT license server. Other methods, such as restricted router interfaces or non-routable IP addresses, might be more appropriate. As a practical matter, we understand that few ISV zPDT users have remote license servers on the general internet.

8.8.2 Resetting UIM

You can usually remove the local UIM serial number with the **`uimreset -l`** command. You can remove both the local UIM serial number and corresponding entries in a remote UIM server database with the **`uimreset -r`** command.

If the local UIM database is corrupted, the **`uimreset`** command might fail. In this rare case, you (working as root) can delete file `/usr/z1090/uim/uimdatabase.db`. However, the previous UIM serial for that client still is provided by a UIM server (if the client is configured for connection to the server). In this case, the **`uimreset -r`** command can be used to remove the relevant entry from the UIM server database if that is wanted. If the `/usr/z1090/uim` directory is deleted, it can be re-created by the following commands:

```
$ su
# cd /usr/z1090
# mkdir uim
# chgrp zpdththru uim                  #This group name is required.
# chmod 2770 uim                       #These permissions are required.
```

A UIM server can be reinitialized by removing everything in the `UIMserver` subdirectory in the home directory of the Linux user ID that runs the UIM server. This step is a drastic one, so it should *not* be done in normal operational environments. If the `UIMserver` directory is cleared, some of the entries will be restored by future client connections in which the client still has previous UIM local data.

The client configuration file can be restored to its original state (which does not reference any remote servers) by using the Factory Reset option with the **`clientconfig`** command.

8.8.3 SafeNet module restarts

There are two SafeNet functions that are involved with ISV zPDT:

- The license servers that we describe in this chapter.
- A daemon (“token driver”) that communicates with Gen1 tokens (in USB ports).

After ISV zPDT is installed, both these functions are started automatically when Linux starts. Changing the license server files requires restarting the license server. It should not be necessary to restart the token driver except in unusual situations.

The commands to restart the Gen1 USB token daemon are as follows:

```
$ su                                   (Change to root.)
# cd /opt/safenet_sentinel/common_files/sentinel_usb_daemon
# ./load_daemon.sh restart             (or status or stop or start)
```

The commands to restart the Gen1 server are as follows:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart                   (or status or stop or start)
```

8.9 Server search

A backup Gen1 server can be specified by a client. The servers are searched for an appropriate license in the order that is listed. There is no coordination among multiple servers, that is, each one must have available licenses to serve them to clients. Either the customer installation purchased more licenses or split the available licenses among multiple servers in some way.

An ISV zPDT client searches all available license sources until it finds the licenses it requires. If remote license servers are defined for a client but cannot be accessed by a TCP/IP connection, there will be delays while the access attempts time out before another license server is tried. If no `Gen1 ContactServer` is specified with the **`clientconfig`** command, ISV zPDT internally specifies `localhost` for this operand. So, if all other license searches fail, ISV zPDT looks for a local Gen1 token in the client system.

8.10 ISV zPDT serial numbers

Consider a remote license server with five ISV zPDT licenses that it can allocate to clients. A single client can request all five licenses by coding `processors 5` in their devmap. Five different clients can each request a single license. There can be a combination of clients that consume the five available licenses. When a client ISV zPDT ends (with the **`awsstop`** command), the licenses that are used by that client are available to other clients. At any given point, no more than five ISV zPDT client licenses, representing five CPs, can be allocated to clients.[18]

Over time, many client ISV zPDT systems might connect to this remote license server if not more than five licenses are allocated at any one time. Each of the many clients has a unique serial number that is provided by the remote UIM server. You might have a case where five licenses are available, but 10 serial numbers are associated with these five licenses. This distinction between numbers of licenses and numbers of serial numbers might be important for some ISV software license situations.[19]

A single ISV zPDT instance cannot have more than eight CPs,[20] each requiring an ISV zPDT license.[21] (IBM contract conditions might specify a smaller limit.) Assuming that the maximum of eight can be used, the devmap for an instance can request eight licenses from the remote server.[22] In our example, only five licenses are available, and the client receives all five licenses (if no one else is using any licenses). Perhaps the intention of the customer is to share their five licenses among several development systems. There is no technical way to prevent a single user (that is, a single ISV zPDT system) from using all the licenses (up to eight, if that many licenses are available). Management control is needed to ensure “fair” sharing of ISV zPDT licenses in situations where a limited number of licenses are serving multiple remote clients.

[18] We ignore differences for specialty engines in this discussion.

[19] There might be more contractual limitations to the number of users (people) or clients (ISV zPDT systems) that are involved.

[20] IBM zSystems Integrated Information Processor (zIIP), IBM zSeries Application Assist Processor (zAAP), and IBM Integrated Facility for Linux (IFL) processors also are counted along with CPs. The total cannot exceed eight.

[21] A PC running ISV zPDT should have more cores than the number of ISV zPDT CPs that is requested by the devmap. In this discussion, we assume that the client has sufficient cores to meet this requirement.

[22] Starting with ISV zPDT GA8, zIIP processors do not require a license. However, they still count toward the maximum of eight CPs in an ISV zPDT instance and toward the number of cores that is needed.

8.11 Token activation and renewal

ISV zPDT Gen1 tokens must be activated (that is, have an initial ISV zPDT license or licenses installed). This task might be done by your ISV zPDT supplier. ISV zPDT token updates are needed to extend the license date (or to replace a corrupted license).

8.11.1 Overview of token updates

ISV zPDT (1090 tokens) has a basic procedure for obtaining token licenses, including license renewals:

1. Use the **`Z1090_token_update`** command to generate a request (`.req`) file based on the token.
2. Send the `.req` file to your ISV zPDT license provider, which can be IBM Resource Link for IBM employees or an external provider for ISVs)
3. The ISV zPDT provider returns, usually by email, a compressed file that is installed with the **`Z1090_token_update`** command. Do *not* decompress the file. Let the command decompress what is necessary.

Older ISV zPDT releases before ISV zPDT GA5 use the **`SecureUpdateUtility`** command to obtain `.req` files or install `.upw` files. This utility is still provided, and instructions for its use appear in earlier ISV zPDT documentation.

The installation of the licenses in the compressed file (with the **`Z1090_token_update`** command) can take longer than you might expect (up to a minute). Be patient when using the command. Remember to unplug a token for 10 - 15 seconds after updating it, which causes Linux to read the new license that you installed.

Multiple tokens

Only one token can be present in a system that is updating the token, and ISV zPDT cannot be active when applying the update. You can create a request file (`.req`) while ISV zPDT is active if only one token is present. If you have multiple tokens, you must update them one at a time. You must be logged in to the machine that has the USB token connected to it to do these activities. You cannot update a token in a remote server.

8.11.2 Token license update details (1090 tokens)

Important: The following details are primarily for IBM internal users. For other users, your ISV zPDT supplier probably handles these details for you.

At the time of writing, a USB hardware key (token) is normally valid for a year from the time it was last activated. Activation (and *lease renewal*) might be handled by your ISV zPDT service provider (such as an IBM Business Partner) or by using IBM Resource Link in some cases.

Copy the information that is printed on the token tag (illustrated in Figure 8-3 on page 197) attached to the USB hardware key.[23] Store this information in a safe place because this information is needed to activate a replacement token if replacement becomes necessary.

[23] 1091 tokens do not have this tag. They have a serial number that is engraved on the back, but it is not used for the process that is described here.

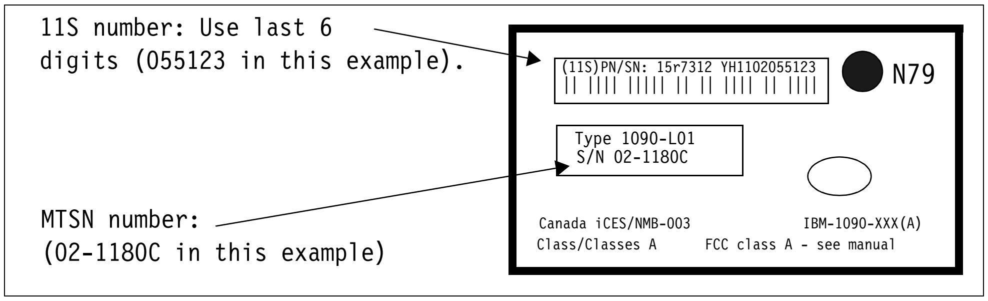

11S number: Use last 6 digits (055123 in this example).
MTSN number: (02-1180C in this example)
(11S)PN/SN: 15r7312 YH1102055123
N79
Type 1090-L01
S/N 02-1180C
Canada iCES/NMB-003
Class/Classes A
IBM-1090-XXX(A)
FCC class A - see manual

Figure 8-3 USB hardware key tag

An IBM Resource Link profile (user ID) is needed[24], which can be an IBM employee profile or a PWD-approved profile for other users. An IBM employee can go to IBM Resource Link and click **Register for an IBM ID** (in the upper right of the page):

After this preliminary work, the token license can be activated, renewed, or have the lease date extended (they are all essentially the same operation):

1. Connect the USB hardware key to your ISV zPDT system by using any USB port. (You must have the ISV zPDT software installed.) Only one token can be connected while doing this task.
2. Working as root, create a request file by using **`Z1090_token_update`**. Log in with a normal ISV zPDT user ID, such as `ibmsys1`.

```
$ su                                  (Change to root.)
# cd /usr/z1090/bin                   (Must be in this directory.)
# Z1090_token_update -r myrequest
# exit                                (Leave root.)
```

3. If necessary, move this request file to the computer that is used to access Resource Link and log on to Resource Link. (Your request file name has `.req` added as the name extension.)
4. On Resource Link, select **Tools** → **1090 Support** → **Date Extension** and enter the data from your hardware key tag. Use the last 6 digits of the 11S field. The serial number (the MTSN field) can be entered with or without the dash (it is not case-sensitive). Enter the file name of your request file. The PC Identifier field is not verified, so as a best practice, enter your PC serial number there. Click **Submit**.
5. Resource Link creates one or two files and sends them to you by email. Receive the files and move them to your ISV zPDT machine, if necessary. The file names are the same name that you sent, but with `.upw` and `.zip` as the name extensions.
6. Apply[25] the file to the Gen1 token:

```
# Z1090_token_update -u myrequest.zip
```

After the update is successfully applied, unplug the USB hardware key. Wait 10 - 15 seconds and then reconnect the hardware key. It is now ready for routine ISV zPDT operation.

[24] For various historical reasons, some IBM employees have customer Resource Link user IDs. These user IDs do not see the required links in Resource Link.

[25] **`SecureUpdateUtility`** is used with ISV zPDT releases before GA5, and **`Z1090_token_update`** is used with ISV zPDT release GA5 and later.

8.12 Unable to use a license

If you cannot use a license when you think that your licenses should work, complete the following steps:

1. Start ISV zPDT. If you receive a message about an expired license, you must obtain a new license update file.
2. If ISV zPDT starts, issue a **`token`** command. You should see the relevant operational license and the expiration dates.
3. Although the function is not a part of the IBM zPDT product, you might try a function that is provided with the SafeNet license functions. Start a browser and point it to port 7002 on the license server. (When working directly on the ISV zPDT system, you might point to `localhost:7002`.) You should see a SafeNet window. Complete the following steps:
   a. On the right side, click **Keys Information**.
   b. Click the number (usually "1") in the Key# column.
   a. The installed keys should appear. The License ID is interpreted as follows:
      i. For ISV zPDT, x'abcd' is the emulation operational key.
      ii. For ISV zPDT, x'd98' and x'8198' are the keys to install ADCD IPL volumes.

8.13 Summary of relevant ISV zPDT commands and files

The following ISV zPDT commands are related to licenses, tokens, license servers, UIM, and UIM servers:

`clientconfig`	Configures both Gen1 and Gen2 client systems.
`clientconfig_cli`	Provides a non-interactive interface to Gen1 clients to configure a connection to a Gen1 license server.
`clientconfig_authority`	Provides a path to use **`clientconfig`** from a non-root Linux user ID.
`request_license`	Generates a file that you send to an IBM authorized facility to obtain a license file for your Gen2 software license server.
`SecureUpdateUtility`	An older utility for updating Gen1 tokens. It is now deprecated.
`SecureUpdate_authority`	Provides a path to use **`SecureUpdateUtility`**, **`Z1090_token_update`**, or **`Z1091_taken_update`** commands from a non-root user ID through the **`zPDTSecureUpdate`** command.
`token`	Displays details of the operational licenses in a client or local ISV zPDT instance.
`uimcheck`	Displays (for a client or local ISV zPDT system) the current UIM status.
`uimreset`	Removes the IBM zSystems system serial number from the local or remote UIM database.
`uimserverstart` and **`uimserverstop`**	Start and stop a remote UIM server.
`Z1090_token_update`	Update licenses in Gen1 tokens.
`zPDTSecureUpdate`	A more convenient way to update Gen1 tokens.

8.14 License manager glossary

The license server and UIM server functions can be confusing. The glossary that is presented here can help with the key terms.

SafeNet	The product line that provides the USB keys and the software that directly supports them, which includes the USB driver, the license manager, and a web interface to the license manager. The original owning company was purchased by Gemalto N.V., which was purchased by Thales or Thales Group. We continue to use the SafeNet name in this chapter although it is considered to be a solution rather than a company name.
SafeNet Sentinel Key	The USB “token” from the SafeNet product line. This token provides ISV zPDT license information. However, a “key” can also be a software-only function of a Gen2 server or a Gen2 hardware token. The terms *token*, *key*, *SafeNet key*, and *Sentinel key* are used interchangeably.
License	A logical function that enables one IBM zSystems CP for an ISV zPDT system. Multiple CPs require multiple licenses. The token functions (or the software-only license function of a Gen2 server) provide licenses.
Heartbeat	The periodic accessing by ISV zPDT of the license (or licenses) that are managed by a license server. If the heartbeat is missed, the zPDT function stops.
Time cheat	The Sentinel Key records the current date and time each time that the key is accessed. If the Linux system clock contains a time earlier that the last recorded time in the token, the license is unusable.
Token serial number	The license information in the token contains a unique serial number that is assigned by IBM. This serial number *might* be used as the basis for the IBM zSystems CP serial number in some cases.
UIM	This is a server (or local function of ISV zPDT) that helps maintain unique, enterprise-wide IBM zSystems serial numbers for ISV zPDT systems. The license server and the UIM server (or local function) are separate but parallel functions.
Identification	A serial number and instance number, as stored by the IBM zSystems system STIDP instruction. (The instance number is like an LPAR number on a larger IBM zSystems server.)
Serial number	A value 1 - 65535 (4 hex digits). The serial number is assigned by the UIM function to the base Linux and used by ISV zPDT to provide the IBM zSystems serial number.

Instance number	A number 1 - 255 that is assigned to each ISV zPDT instance on a base Linux machine. Each ISV zPDT instance must operate under a different Linux user ID, and the instance number is assigned to the user ID. The instance number is used in the same manner as the LPAR number on a larger IBM zSystems server.
UIM database	A file containing UIM information. The files are not directly editable. There are two types of databases. One exists in every Linux ISV zPDT machine, and the other exists in a UIM server (if it is used). The local database (on an ISV zPDT client) is at this location: `/usr/z1090/uim/uimclient.db`
Random serial number	A serial number that is unique but is not tied to a token serial number. The UIM server generates and assigns these numbers. A random serial number can be used (by ISV zPDT) with a license from any token. (Do not take the “random” word too literally. In this case, “random” means that tokens with serial numbers other than the one that is used to set the UIM serial number might be used. It does not mean that you can select a random number.)
IBM Rational® License Server	No relation to ISV zPDT license servers. It provides controlled access to multiple IBM software products, and it might be used with IBM ZD&T license servers.

9

Other IBM zSystems operating systems

This publication is primarily concerned with z/OS. However, other IBM zSystems operating systems can be used under Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT), assuming that the relevant software license permits it. This chapter provides a brief overview and assumes that potential users of these operating systems have relevant experience.

9.1 z/VSE

z/VSE is a possible operating system to run under ISV zPDT. For more information about licensing and downloading, consult your ISV zPDT provider.

9.2 Linux for IBM zSystems

Linux for IBM zSystems (or a similar name) is not an IBM product and not distributed by IBM. Several Linux products and distributions can be considered under the generic name of Linux for IBM zSystems or Linux S390X. There is no equivalent of an Application Development Controlled Distribution (ADCD) that is available for Linux for IBM zSystems.

Various installation procedures might be used for these products, but we do not attempt to cover the techniques in this publication. The ISV zPDT **`ipl_dvd`** command might be relevant if your Linux for IBM zSystems distribution is on a DVD and the DVD is configured for direct installation. Some versions or updates of Linux for IBM zSystems can include changes that affect the installation process. The zPDT developers sometimes cannot predict these changes ahead of time, so you might need to search for workarounds.

9.3 z/VM

Important: ISV zPDT GA11 is at the IBM Z16 architecture level. z/VM 7.1 and 7.2 require specific service for usage at the z16 architecture level, and ADCD z/VM 7.2 does not have this service applied.

At the time of writing, any updating of ADCD z/VM 7.2 is uncertain.

Some uses of ADCD z/VM 7.2 appear to be usable with zPDT GA11, but a few specific functions (such as ZCMS for IBM z/CMS) have immediate problems.

If you have important work requiring z/VM usage, you might consider delaying usage of ISV zPDT GA11 until another ADCD release is available. Another option is to apply the required service to z/VM 7.2 if you have access to the service materials.

The text in this chapter assumes the usage of the z/VM 7.2 ADCD and the limitations that are mentioned here.

z/VM is available in an ADCD format for ISV zPDT users whose license includes access to z/VM. However, a separate license agreement *might* be required.

The ISV zPDT GA11 release corresponds to the z16 architecture. At the time of writing, z/VM 7.2 was the latest z/VM ADCD package that is available to authorized users and suitable for use with ISV zPDT GA11. Older releases of z/VM might not work with z16 systems or might require specific service levels or specific customization to work with the most recent IBM zSystems servers. Older editions of this publication address some of these considerations. This section deals only with ISV zPDT GA11 and the ADCD z/VM 7.2 release.

In particular, the parameter **“`PAGING63`”** (which was described at length in previous editions of this publication) is not relevant when using ISV zPDT GA10 (or later) and z/VM 7.1 or z/VM 7.2. (z/VM release 7.1 might be the last release to support it.)

A common usage of z/VM with ISV zPDT is to create a z/OS Parallel Sysplex environment. For more information, see *zPDT Sysplex Extensions - 2020*, SG24-8386.

At the time of writing, the ADCD z/VM 7.2 system is not preconfigured for TCP/IP operation. Any networking is up to the user to configure.

z/VM now includes two versions of Conversational Monitor System (CMS): CMS and ZCMS. The traditional CMS uses the z/Architecture S/390 Compatibility Mode for operation, which has some restrictions, as noted in “S/390 Compatibility Mode” on page 274. ZCMS is a more modern, fully z/Architecture version. It is started with the z/VM command **`IPL ZCMS`**.

Tip: zPDT is not intended for production usage or to fully duplicate the operation of a “normal” IBM zSystems system. General restrictions are listed in Chapter 2, “ISV IBM Z Program Development Tool concepts and terminology” on page 19. In addition to these listed restrictions, there can be limitations on attempts to run unusually large applications on a zPDT system. One such limitation is the size of Coupling Facility (CF) vector tables. These tables are limited to 16 pages (or 524,288 bits) as provided through the z/VM CF function on a zPDT system. We expect that this limitation rarely will be encountered for reasonable ISV development and testing, but it might be seen with unusually large (for a zPDT system) Db2 buffer settings.

9.3.1 Installing the ADCD z/VM 7.2 system

At the time of writing, six 3390 volumes are used for the ADCD z/VM 7.2 system. These volumes are as follows:

- M01RES
- VMCOM1
- 720RL1
- 720RL2
- M01P01
- M01S01

As with other ADCD IBM zSystems volumes, these volumes are packaged as compressed (**`gzip`**) files and are expanded into usable form with a **`gunzip`** command, as in this example:

```
$ gunzip -c m01res.gz > /z/M01RES
```

This example assumes that the `gz` file is in the current Linux directory and the target directory for the volume is in the `/z` directory. These locations are arbitrary, and you must adapt the commands to your situation. ADCD distribution files often have lowercase names (`m01res.gz`). Our examples expand the files with uppercase names (`M01RES`), but uppercase is not required. (We do it to help distinguish emulated 3390 volumes in the Linux directory.)

With the early ADCD z/VM 7.2 system, these volumes (once expanded with **`gunzip`**) contained 9.3 GB sizes instead of the 8.5 GB sizes of previous ADCD z/VM volumes, representing about 11032 cylinders instead of the traditional zPDT 3390-9 with 10017 cylinders.

At the time of writing, the ADCD z/VM initial program load (IPL) volume is not encrypted, and it is installed with a simple **`gunzip`** command. Later z/VM releases might have an encrypted IPL volume like the one that is used with current ADCD z/OS releases.

9.3.2 ISV zPDT device map for ADCD z/VM 7.2

We created a devmap that is named `devmapvm72` that defines a minimal z/VM system. The addresses in this example differ from the ones for previous releases and are required, at least, for initial usage of the system.

```
[system]
memory 8000m
3270port 3270
processors 3

[manager]
name aws3274 0002
device 0700 3279 3274
device 0701 3279 3274
device 0702 3279 3274
(We usually define at least ten 3270 sessions)

[manager]
name awsckd 0101
device 0123 3390 3990 /z/M01RES
device 0124 3390 3990 /z/VMCOM1      (Address of VMCOM1 is important.)
device 0125 3390 3990 /z/720RL1
device 0126 3390 3990 /z/720RL2
device 0127 3390 3990 /z/M01S01
device 0128 3390 3990 /z/M01P01
```

Other disks, LAN interfaces, and tape drives might also be defined.

The ADCD z/OS systems (at the time of writing) define disk addresses 120 - 15F as 3380 devices and not as 3390 devices. (They are defined in the z/OS input/output definition file (IODF) that is used.) This difference might create a slight challenge if you use the same devmap for z/OS and z/VM 7.2.

9.3.3 z/VM IPL and logon

Based on the sample devmap that is shown in 9.3.2, "ISV zPDT device map for ADCD z/VM 7.2" on page 204, we used the following ISV zPDT command to perform an IPL of this system:

```
$ ipl 120 parm 0700
```

This command opens the stand-alone loader panel that is shown in Figure 9-1.

```
STAND ALONE PROGRAM LOADER: z/VM VERSION 7 RELEASE 2.0
DEVICE NUMBER:   0123 MINIDISK OFFSET:       39          EXTENT:  1
MODULE NAME:     CPLOAD     LOAD ORIGIN:       1000
--------------------------------IPL PARAMETERS--------------------------------
fn=SYSTEM ft=CONFIG pdnum=1 pdvol=0124 cons=0700
-----------------------------------COMMENTS-----------------------------------

9= FILELIST  10= LOAD  11= TOGGLE EXTENT/OFFSET
```

Figure 9-1 Stand-alone loader panel

We added the `cons=0700` section in the `IPL PARAMETERS` line to use the 3270 session at address 700 as the z/VM OPERATOR session. The first start might require the **`FORCE`** option to continue with the IPL, but this approach might change in later ADCD z/VM releases. (The 700 address is a not required value, but it is convenient because it is the “normal” z/OS operator address for ADCD z/OS, so we automatically add it to most devmaps.)

During z/VM startup, you can perform a “warm start” or “cold start” the system. As a best practice, use warm start unless you have a specific reason for cold starting z/VM. You might need to reply `WARM` at an operator prompt and `NO` for a prompt about the time-of-day (TOD) clock. Typically, pressing Enter is all that is needed for these prompts.

The `CPREAD` at the bottom of Figure 9-2 means that z/VM is waiting for your input. During a start, you can press the 3270 Enter key. “`MORE...`” or “`HOLDING`” at the bottom of a panel indicates that you should clear the panel.[1] You might need to clear it several times until all the initial OPERATOR messages are displayed. The “`IBMSYS1`” at the lower right of the panel is the z/VM system name.

```
 10:13:24 z/VM  V7 R2.0  SERVICE LEVEL 2001 (64-BIT)
 10:13:24 SYSTEM NUCLEUS CREATED ON 2020-07-29 AT 16:50:40, LOADED FROM M01RES
10:13:24
10:13:24 ****************************************************************
10:13:24 * LICENSED MATERIALS - PROPERTY OF IBM*                        *
10:13:24 *                                                              *
10:13:24 * 5741-A07 (C) COPYRIGHT IBM CORP. 1983, 2018. ALL RIGHTS      *
10:13:24 * RESERVED. US GOVERNMENT USERS RESTRICTED RIGHTS - USE,       *
10:13:24 * DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE     *
10:13:24 * CONTRACT WITH IBM CORP.                                      *
10:13:24 *                                                              *
10:13:24 * * TRADEMARK OF INTERNATIONAL BUSINESS MACHINES.              *
10:13:24 ****************************************************************
10:13:24
10:13:24 HCPZCO6718I Using parm disk 1 on volume VMCOM1 (device 0207).
10:13:24 HCPZCO6718I Parm disk resides on cylinders 1 through 120.
10:13:24  Start (Warm|Force|COLDCLEAN) (DRain) (DIsable) (NODIReect)
         (NOAUTOlog) or (SHUTDOWN)
                                                       CPREAD IBMSYS1
```

Figure 9-2 Initial OPERATOR display (older z/VM version)

[1] ISV zPDT users of x3270 often set the PAUSE or END key (on their personal computer (PC) keyboard) to perform a 3270 clear operation. If this action is not done, then Alt+C might be used for the clear function with some 3270 emulator setups.

Now, you should see the z/VM logo that is displayed on any other active 3270 sessions, as shown in Figure 9-3. If the logo display is not present, try running an **ENABLE ALL** command in the OPERATOR session.

```
z/VM ONLINE

                                / VV          VVV MM       MM
                               /  VV        VVV  MMM     MMM
                   ZZZZZZ     /   VV      VVV   MMMM   MMMM
                      ZZ     /    VV    VVV    MM MM MM MM
                     ZZ     /     VV  VVV     MM  MMM  MM
                    ZZ     /      VVVVV      MM   M   MM
                   ZZ     /       VVV       MM       MM
                  ZZZZZZ /        V        MM       MM

                    built on IBM Virtualization Technology


Fill in your USERID and PASSWORD and press ENTER
(Your password will not appear when you type it)
USERID   ===>
PASSWORD ===>
COMMAND  ===>
                                                          RUNNING IBMSYS1
```

Figure 9-3 z/VM logo display

You can disconnect from the OPERATOR session by running a **DISC** command. This action is not the same as logging off from the session. A disconnected session continues to run, but without the terminal. You can log back into the session. For the operator session, the user ID is OPERATOR.

During the initial panels during an IPL, you see messages about `EREP`, `DISKACNT`, and `OPERSYMP` files displaying the percentages full for each file. In general, z/VM manages these files and you do not need to manage them manually.

9.3.4 z/VM 7.2 initial use

z/VM 7.1 and 7.2 have an interesting method of using passwords. Practically all of the "system" user IDs reject direct logins. For example, you cannot directly log in to z/VM user ID `MAINT`.

Instead, you must use **LOGON MAINT BY IBMVM1**, which is entered in the `COMMAND ===>` line in the normal z/VM 3270 login panel. The distributed password for user IBMVM1 is `ZVM720`, which also is the password for those guest user IDs that do accept "normal" logins. (This password might change in later releases, but the technique is likely to remain.)

A guest that is named `IBMUSER` is defined with password `WD5JU8QP` or `ZVM720`. It has a small A disk, but can be used as a non-privileged user. (This `WD5JU8QP` password for a number of user IDs appeared in the ADCD z/VM 7.2 version that was available at the time of writing. If you plan on serious ADCD z/VM use, consider working with the `USER DIRECT C` file to resolve any password concerns, and then use the **DIRECTXA** command to activate the changed file.)

Changing the stand-alone loader parameters

The stand-alone loader IPL parameters can be changed by logging in to z/VM as user ID MAINT720 (we used the password ZVM720, but you might need WD5JU8QP instead) and entering the following command:[2]

```
SALIPL 123 (EXTENT 1 IPLPARMS fn=SYSTEM ft=CONFIG pdnum=1 pdvol=0124 cons=0700
```

Once this command is run, it is no longer necessary to include the **“parm 0700”** option in the zPDT **ipl** command. (This **SALIPL** command also can be used for more changes that are not covered in this publication.)

Tip: As a best practice, do not use the **SAIPL** function until you are familiar with z/VM usage and the z/VM editor. Be careful with this command.

The **pdvol=** parameter points to the address of VMCOM1 volume, but this detail might change in future releases. The address also is relevant to the zPDT devmap that is used.

Parallel Sysplex setup

Several previous ADCD z/VM releases contained z/VM “guest” user IDs that were convenient for z/OS Parallel Sysplex operation. These guest user IDs included MVSDUMMY, BASEAD, SOW1, SOW2, CFCC1, CFCC2, and CFCONSOL. The early version of ADCD z/VM 7.2 that we used at the time of writing did not have these guest user IDs defined. A later ADCD z/VM 7.2 version might include them.

z/VM shutdown

To stop z/VM, issue the **SHUTDOWN** command in the OPERATOR session or in another session that has sufficient privileges.

9.4 Brief z/VM review

We assume that someone wanting to use z/VM on their ISV zPDT system already has some z/VM experience. The following sections are intended as reminders for readers who have not used z/VM for a while.

9.4.1 CMS

Basic CMS usage is required for almost any administrative activity for a z/VM system. It is possible to use CMS for application development and as a base for user applications. This usage is not described here.

After performing an IPL on z/VM and logging on with a user ID, you must perform an IPL for CMS with that user ID (assuming that you want to use CMS). This CMS IPL can be automatic (when you log on to the user ID) or it can be done manually with an **IPL CMS** or **IPL 190** command (or **ipl zcms** for the newer version of CMS.) A RUNNING indicator in the lower right of the 3270 panel is an indicator that CMS is running.

In general, both CMS commands and CP commands can be entered on the CMS CLI. In a few cases, CP commands must be prefixed by the letters CP or #CP followed by a space and then the command.

[2] The WD5JU8QP password might be unique to the particular ADCD z/VM version in use at the time of writing.

When using CMS (or interacting directly with CP if CMS has not been started), the 3270 session is in a pseudo-3215 mode. This mode is a typewriter-like interface, with a CLI at the bottom of the panel. Some CMS functions, such as **`XEDIT`**, provide a full-panel interface like the IBM Interactive System Productivity Facility (ISPF).

9.4.2 User MAINT

z/VM has pre-defined user IDs `MAINT` and `MAINT720`, and most basic z/VM administration is performed while using these user IDs and working through CMS. (The `MAINT720` user ID is likely to change in newer releases, but the general technique will probably remain.) The terms *z/VM users*, *guests*, *user IDs*, and *virtual machines* (VMs) are used interchangeably.

9.4.3 Mini-disks and files

z/VM works with three types of IBM zSystems system disk volumes:

- CP-owned volumes: They contain the mini-disks that are needed to run z/VM, or contain paging space, spool space, or temporary space. They might also contain user mini-disks.
- User-owned volumes: Typically, they contain mini-disks, but not the ones that are used directly by the operating system.
- Other disks: They can be used as standard (non z/VM) volumes or as whole-pack mini-disks. The z/OS volumes that are sometimes used under z/VM are normally defined as whole-pack mini-disks. The whole-pack mini-disk concept is needed for sharing volumes among multiple z/VM VMs, such as multiple z/OS systems.

While the outline is simple, the effective usage of the various disk types can be more complicated, which is an area where prior experience is useful.

To access a mini-disk under CMS, an access mode letter must be assigned. For example, if you have mini-disk 456 that is defined in your z/VM directory entry, you might enter the CMS command **`acc 456 z`** to access the mini-disk as the “z” disk. Your 191 disk (which is almost always defined for a CMS user) is your default “a” disk. For some functions, your mini-disks are searched in order, that is, “a”, “b”, “c”, and so forth. The mode letter (the “disk letter”) is not a fixed value. You might use **`acc 456 r`** the next time that you log on and access your 456 mini-disk as your “r” disk.

9.4.4 Your disks

You can list the files on a mini-disk with a command such as **`filelist * * a`**. This example lists all file names (first asterisk) and all file types (second asterisk) on the mini-disk that are accessed as drive “a”.

When a new CMS disk exists (perhaps with a new z/VM user ID), it must be formatted if it is going to be used for CMS. Assuming the new disk is a 191 disk (the default “a” disk for a CMS user), you can issue the command **`format 191 a`**, reply `1` to continue, and then reply `Bill91` (or some other 6 characters) to specify the mini-disk label.

Another user’s mini-disk

You might want to access another user’s mini-disk (if it is not password-protected). If user Joe has mini-disk 456 defined, you might do the following tasks:

```
link joe 456 456              (Link to joe’s 456 as your own 456.)
acc 456 j                     (Access the disk as your j disk.)
filelist * * j                (See what files are on the mini-disk.)
```

There are two minor problems with this method:

- What if there is a mini-disk that already is defined at address 456?[3]
- What if you are already using file mode `j`?

If either of these items are true, you can use this method instead:

```
vmlink joe 456                 (Use the vmlink command instead.)
DMSVML2060I JOE 456 linked as 0120 file mode z
Ready
filelist * * z
```

In this case, z/VM selected an unused address (0120) and an unused file mode (z) for you. When you are finished with Joe's 456 mini-disk, use the following commands:

```
rel z                          (Free file mode z.)
det 120                        (Detach the mini-disk from your session.)
```

9.4.5 XEDIT

XEDIT is the normal CMS editor. Most z/VM administration and customization are done by editing CMS files. You can create CMS files with XEDIT by naming it. For example, **`x newfile stuff a`** creates a file named `newfile stuff` and places it on your "a" disk (assuming you save the file before exiting from XEDIT).

Important XEDIT commands (on the CLI) are as follows:

`file`	Save the current file.
`qquit`	Exit without saving the file.
`/xxx/`	Find the next line containing characters xxx.
`+nn`	Scroll forward nn lines.
`-nn`	Scroll backward nn lines.
`top`	Scroll to the beginning of the file.
`bottom`	Scroll to the end of the file.
`nulls on` or **`nulls off`**	Select 3270 nulls or blanks usage.

The default PF key definitions are as follows:

- PF1: Help.
- PF2: Insert a blank line after the line with the cursor.
- PF3: Quit (if no changes were made) or exit from the Help function.
- PF4: Use the Tab key to go to columns 5, 10, 15, and so forth.
- PF7: Scroll backward.
- PF8: Scroll forward.
- PF9: Split or Join, depending on the cursor location.
- PF10: Scroll right 10 columns.
- PF11: Scroll left 10 columns.
- PF12: Issue a **`file`** command.

[3] You can use a different address, such as `LINK JOE 456 789`, but then you must determine whether you already have a 789 disk. This problem is a trivial one for most users, who have few mini-disks that are defined, but it might be a significant problem for some users.

Several XEDIT line commands (typing over the `=====` field in the window) are as follows:

d or **d*nn***	Delete one or *nn* lines.
i or **i*nn***	Insert one or nn lines.
" or **"*nn***	Repeat the line one or nn times.
dd followed by **dd** in a later line	Delete the indicated block of lines.
cc followed by **cc** in a later line	Copy the indicated block. The target is noted with `p` (before this line) or `f` (following this line).

XEDIT has many more commands and facilities than are mentioned here, but these few commands might be sufficient for initial usage.

9.4.6 z/VM directory

z/VM users and mini-disks are defined in the z/VM directory. There are two forms of the directory: the *source file*[4] and the *active directory*. The **directxa** command reads the source file and creates (or updates) the active directory.

If you log on as `MAINT`, you can access the source directory with one of these commands:[5]

```
browse user direct c        (Browse it.)
x user direct c             (Edit it.)
```

z/VM might contain the DIRMAINT tool that is typically used to maintain the z/VM directory. For small z/VM systems, such as a sandbox ISV zPDT system, you can "manually" work with the directory.[6] As a best practice, browse the directory on your z/VM system to look at it. A quick look includes the following items:

- The `PROFILE` stanzas define lines that can be included in user definitions by using an **INCLUDE** statement.
- Skip over sections such as `USER $DIRECT$ NOLOG`. They help produce a clean disk map.
- The first line of a "normal" user ID definition begins with the keyword **IDENTITY** or **USER**. In a simple environment, the **IDENTITY** statement is equivalent to a **USER** statement.
- Ignore the **SUBCONFIG** statements, but observe that many statements are "commented out" by an asterisk in the first column.
- **LINK** statements refer to a mini-disk that is owned by another user. For example, in the `MAINT` directory definitions, `LINK PMAINT 2CC 2CC MR` means that the mini-disk at address 2CC in the user `PMAINT` definitions is used at address 2CC in the `MAINT` VM.

A simple user definition might be added, as shown in Figure 9-4 on page 211 (these lines are added to the directory between two existing user entries, or at the end).

[4] There might be multiple source files, but we ignore this detail here.

[5] By default, the source file is named `USER DIRECT`, and the logon profile for `MAINT` accesses the volume as "c".

[6] Manually editing the z/VM directory is not a recommended process for serious z/VM users. However, it does bypass the learning curve for using DIRMAINT.

```
**************************************************
* USER BILL IS TO DEMONSTRATE A SIMPLE VM USERID
*
USER BILL W2WO 128M 128M G
MACH ESA
CPU 0
IPL 190
SPOOL 00C 2540 READER *          Do not copy this
SPOOL 00E 1403 A                 statement.
CONSOLE 009 3215 T
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR
LINK TCPMAINT 592 592 RR
MDISK 191 3390 6059 10 VMCOM1 MR
*
```
Figure 9-4 Simple user definition

In this example, the user ID is `BILL` and the password is `W2WO`. The VM is 128 MB, which is more than ample for CMS. The **`IPL`** statement causes an automatic IPL of CMS when user `BILL` logs on to z/VM. The **`SPOOL`** statements are probably not used for simple situations, but are traditional. A console is necessary, and 009 is a traditional address. The **`LINK`** statements point to mini-disks of other users (all in read-only mode). The specific **`LINK`** statements that are shown here typically provide basic CMS functions.

The **`MDISK`** statement defines a mini-disk that is owned by this user. This example specifies that the mini-disk is placed on a 3390 with volser VMCOM1, with the mini-disk starting on cylinder 6059 and going on for 10 cylinders. The **`MR`** operand specifies basic read/write access. Do *not* copy these exact definitions. Creating a mini-disk (without using DIRMAINT) involves carefully analyzing free space on volumes that are owned by z/VM, and an error can destroy some other user's mini-disks. (If you are relatively new to z/VM and must create mini-disks, consult with a more experienced z/VM user.)

Any editing changes to **`user direct c`** are not used by z/VM until the **`directxa`** command is used. For example, it is commonly **`directxa user direct c`**.

9.4.7 Spool contents

z/VM can emulate card readers, line printers, and card punches for users. Although the equivalent "real" devices are no longer used, these virtual devices can be useful. One way to view the contents of the virtual card reader and printer queue is with the commands **`q rdr`**, **`q prt`**, and **`p pun`**.[7]

You can list the spooled `rdr` files that you own with the **`rdrlist`** command. You can transfer a spool file from another owner to your own rdr, where you can view (**`peek`** command or PF11) it or discard it. The following example transfers OPERATOR's reader file number 20 (the file number from the **`q rdr`** command) to the current (asterisk) user's reader:

```
cp transfer operator rdr 20 to * rdr
```

[7] Each user potentially has their own virtual reader, printer, and punch data.

On a larger scale, the following commands can be used by an authorized user (such as `MAINT`) to purge many or all spooled files:

`purge <userid> rdr <file number>`	Purge a specific rdr spool file.
`purge <userid> rdr ALL`	Purge all the user's rdr files.
`purge <userid> prt <file number>`	Purge a specific prt spool file.
`purge <userid> prt ALL`	Purge all the user's prt files.
`purge system rdr ALL`	Purge all the rdr files in the spool.
`purge system prt ALL`	Purge all the prt files in the spool.

The `<userid>` can be an asterisk, in which case the command applies to the current user. The `<file number>` is usually taken from the **`q rdr`** or **`q prt`** commands.

9.4.8 Simple system queries

A number of simple query, display, and access commands might be useful:

- **`q disk`**: List your mini-disks.
- **`q da all`**: List the "real" online disks.
- **`q alloc all`**: List page, spool, temporary disks, and directory usage.
- **`q alloc map`**: List percentage use for page and spool disks.
- **`q system`**: Another way to list system disks. This command is limited to certain user IDs.
- **`q accessed`**: List mini-disks that you accessed.
- **`q links 120`**: List user IDs who have links to your 120 disk.
- **`q pf`**: List Program Function Key assignments.
- **`q stor`**: List how much storage is on the IBM zSystems server.
- **`q n`**: List which user IDs are logged on.
- **`q all`**: List what disks and terminals are online.
- **`q rdr`**: List the files in your virtual reader.
- **`q prt`**: List the files in your virtual print queue.
- **`set pf12 retrieve`**: Make the PF12 function a retrieve key.
- **`force userid`**: Terminate a user immediately.
- **`rdrlist`**: List files in your virtual reader:
  - Use PF11 to **`peek`** at (view) any of these files.
  - Use **`discard`** to delete a particular file.
- **`purge system rdr all`**: Purge all reader files in the z/VM system:
  - **`purge joe rdr 1234`**: Purge a particular reader file belonging to user ID `joe`.
  - **`purge joe rdr all`**: Purge all of user joe's reader files.
- **`purge system prt all`**: Purge all printer files in the z/VM system.
- **`link joe 456 456`**: Link to joe's 456 disk as your 456 disk.
- **`acc 456 j`**: Access your 456 disk as CMS drive "j".
- **`filelist * * a`**: List the files on your "a" disk.
- **`rel j`**: Release a CMS drive assignment.
- **`det 456`**: Detach disk 456 from your user ID.
- **`vmlink joe 345`**: A combined **`link`** and **`acc`** function.
- **`format 191 a`**: Format a new mini-disk.
- **`directxa user direct c`**: Activate an updated z/VM directory.
- **`ind`**: See how busy the system is.
- **`diskmap user direct c`**: Create a mini-disk map that is based in directory `user direct c`.
- **`browse user diskmap a`**: Inspect a file.

9.4.9 IBM zSystems Integrated Information Processor and IBM zSeries Application Assist Processor processors

z/VM can provide *simulated* IBM zSystems Integrated Information Processor (zIIP) processors, working only with CPs in the base ISV zPDT system. Furthermore, z/VM can provide more logical CPs and zIIPs than there are CPs present in the base ISV zPDT system.

The definition of a z/VM guest or user ID, in the z/VM directory, can contain statements such as the following ones:

```
MACH ESA  5              (Allow up to five logical processors.)
CPU 0 BASE
COMMAND DEFINE CPU 1 TYPE ZIIP
```

The two logical processors (one CP and one zIIP) can be used even if the base ISV zPDT definition has only a single CP (a model1090-L01, for example). There are performance implications if the number of logical processors greatly exceeds the number of “real” IBM zSystems processors, but this situation might be acceptable for development and testing situations.

9.4.10 Paging

If you run z/OS (or another IBM zSystems operating system) under z/VM on an ISV zPDT base machine, you potentially have three levels of paging:

- The base Linux system pages whenever virtual memory usage exceeds the available real memory. As a best practice, have considerably more real memory available than is defined for the IBM zSystems memory size.[8] This allocation minimizes Linux paging. A Linux page fault in the primary ISV zPDT CP process causes the CP to pause until the page fault is resolved. (The criteria for this consideration is more complex than stated here, and partly revolves around the base Linux disk cache usage.)
- z/VM pages when its requirement for virtual memory exceeds the defined IBM zSystems memory. Each z/VM guest (whether a CMS user or a whole z/OS system) is in z/VM virtual memory. z/VM systems on larger machines, running multiple significant guests, tend to page rather heavily.
- z/OS (or another IBM zSystems operating system) pages when its need for real memory (which is z/VM virtual memory) exceeds whatever size was defined in the z/VM directory for the z/OS guest.

An ISV zPDT system has limited I/O bandwidth to its disk, and that bandwidth is best used for running applications rather than for paging. As a best practice, consider your memory usage carefully when you plan to use z/VM for multiple guests.

9.4.11 Starting z/OS under z/VM

It is possible to run one or more z/OS systems as “guests” of z/VM. Assuming that the z/OS disks (in their normal “native z/OS” format)[9] are included in the devmap, the following z/VM commands can be used to start the z/OS IPL process:

```
term conmode 3270
i a80 loadp 0a8200
```

[8] There is no specific formula for this memory management. A fairly common technique is to have available PC memory that is at least twice the size of the zPDT memory that is defined in the devmap, but the effectiveness of this approach depends on the z/OS workload and workload timing that is involved.

[9] This statement implies that the z/OS volumes are not in the z/VM mini-disk format.

The first command causes the z/VM panel to change into the “full screen” mode that is used by z/OS. The second command is an **IPL** command, which in this example assumes that the z/OS system residence volume is at address a80 and that the other initial volume that is needed is at address a82.

In some ways, running z/OS under z/VM is easy, but in other ways (such as defining internet or TCP/IP usage), it can be more complex.

9.5 IBM z/Transaction Processing Facility

IBM z/Transaction Processing Facility (z/TPF) is a specialized IBM operating system that is used for high transaction-rate, high availability applications, such as airline reservations, hotel reservations, and ATM transactions. There is a substantial, well-established user z/TPF community, although their z/TPF usage is often described under their own in-house system names, and “z/TPF” is seldom mentioned publicly. zPDT can be a useful base for z/TPF application development, unit testing, and education. At the time of writing, z/TPF for zPDT licensing is a limited offering. Serious inquiries about z/TPF should go to `TPFQA@us.ibm.com`.

There is no prepackaged z/TPF system like the ADCD packages for z/OS or z/VM. The assumption is that z/TPF customers either generate a suitable z/TPF or migrate one of their working z/TPF systems from another platform to zPDT.

10

Multiple instances and guests

Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) supports both guest operations (under z/VM) and multiple instances of ISV zPDT. Both approaches are a way to run multiple z/OS or other operating systems.

In this chapter, we present basic information about the usage of multiple ISV zPDT instances. Practical operation with multiple instances might be complex. You might need to work with your ISV zPDT provider to clarify usage of more complex configurations.

Tip: For newer ISV zPDT users, as a best practice, first work with a basic system (not in virtual, container, or multiple instances environments) to become familiar with routine operations before venturing into more complex environments.

10.1 Multiple instances or multiple guests

As a best practice, use a single ISV zPDT instance in native mode to become familiar with basic ISV zPDT operation. Avoid running z/OS (or other IBM zSystems operating systems) under z/VM or in multiple ISV zPDT instances until you are comfortable with basic ISV zPDT usage.

Also, you cannot exceed the number of ISV zPDT licenses in your token (tokens or license server). The total number of ISV zPDT licenses applies whether the ISV zPDT CPs are in a single instance or spread over multiple instances.

Multiple *instances mean* running more than one copy of ISV zPDT. Each instance must run under a different Linux user ID. This task can be accomplished by logins through Telnet (or SSH) or by careful usage of `su` commands from different windows on the Linux desktop. Each instance must have its own device map (devmap).

The usage of TCP/IP interfaces is an essential part of this discussion. For this reason, we combine a discussion of multiple ISV zPDT instances with the usage of guests under z/VM in a single instance. Our examples use OSA-Express Direct (OSD) (QDIO) interfaces for TCP/IP. Using OSA-Express (OSE) interfaces (non-QDIO) might not be possible because you must configure the Open Systems Adapter (OSA) Address Table (OAT) by using the Open Systems Adapter/Support Facility (OSA/SF) utility, and this utility is no longer provided with z/OS.

10.2 Multiple guests in one instance

A typical z/VM configuration is outlined in Figure 10-1. z/VM itself typically "owns" all the 3270 sessions. Guests (z/OS and Conversational Monitor System (CMS)) acquire a 3270 when a user logs on as a guest or uses a **`DIAL`** command.

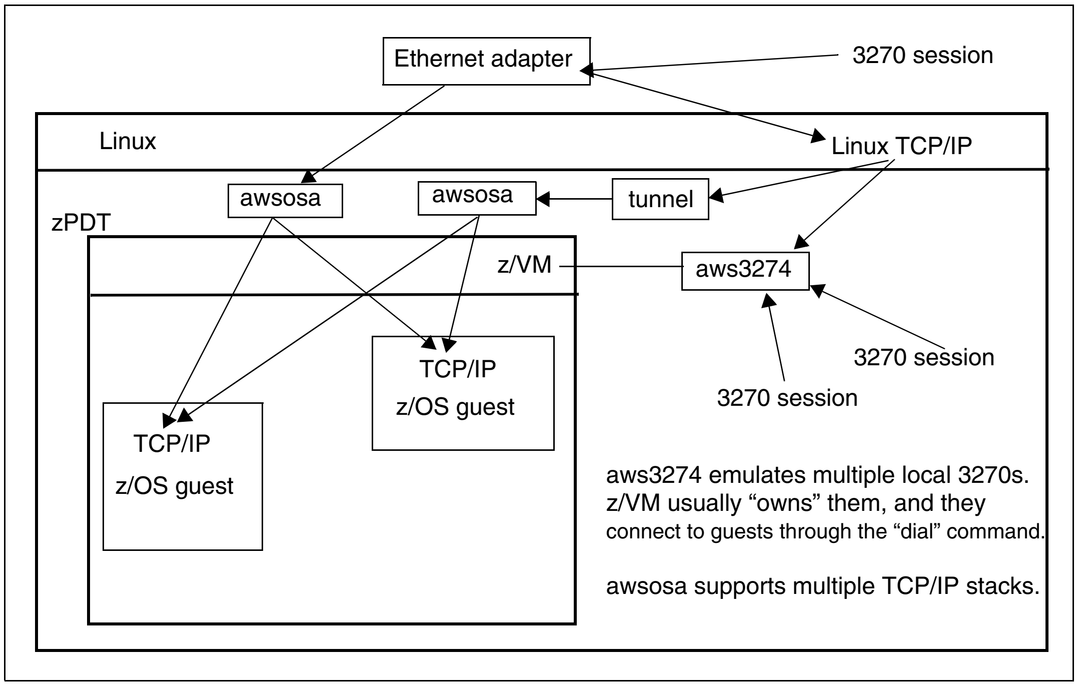

Ethernet adapter
3270 session
Linux
Linux TCP/IP
awsosa
awsosa
tunnel
zPDT
z/VM
aws3274
TCP/IP
z/OS guest
3270 session
3270 session
TCP/IP
z/OS guest
aws3274 emulates multiple local 3270s.
z/VM usually "owns" them, and they
connect to guests through the "dial" command.
awsosa supports multiple TCP/IP stacks.

Figure 10-1 Guests in a single ISV zPDT instance

Each guest (under z/VM) can access a LAN interface. The awsosa device manager can handle up to 16 of these “stacks”. An awsosa device manager that uses a tunnel to Linux can be used, as shown in Figure 10-1 on page 216. Each z/VM guest uses different IP addresses on each OSA interface. Alternatively, which is not shown in Figure 10-1 on page 216, z/VM can establish an internal VSWITCH for guest use. Using the IP address patterns from our other examples, we might have the following addresses in Figure 10-1 on page 216:

`192.168.1.80`	Linux IP address for the Ethernet adapter
`192.168.1.80 port 3270`	Address for external TN3270e connections to aws3274
`10.1.1.1`	Linux IP address for the tunnel interface
`192.168.1.81`	z/VM IP address for Ethernet
`10.1.1.2`	z/VM IP address for the tunnel interface
`192.168.1.82`	z/OS #1 address for Ethernet
`10.1.1.3`	z/OS #1 address for the tunnel interface
`192.168.1.83`	z/OS #2 address for Ethernet
`10.1.1.4`	z/OS #2 address for the tunnel interface
`127.0.0.1`	Localhost connection for local x3270 sessions

10.3 Independent instances

We can have two independent ISV zPDT instances, meaning that emulated I/O devices are not shared between the instances. In common terms, there is no shared direct access storage device (or any other shared device).

Figure 10-2 shows an example of independent instances.

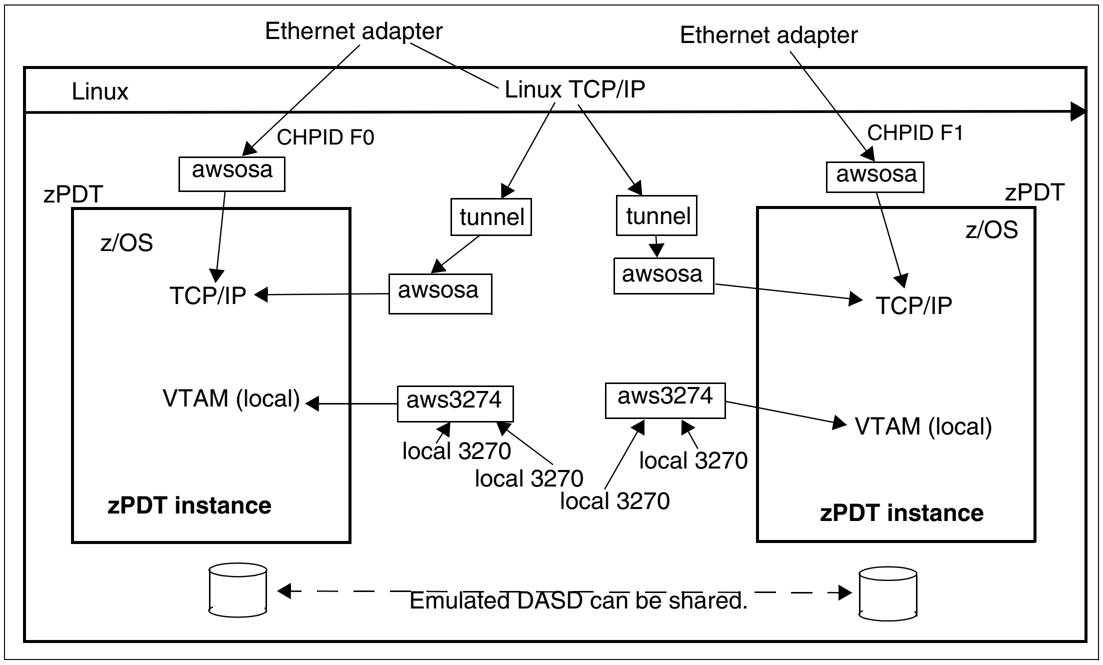


Figure 10-2 Independent instances

In this example, we assume that there are three ISV zPDT licenses (and a base Linux machine with at least four cores), and we assigned two CPs to one instance and one CP to the other instance. Different port numbers are needed in the **3270port** statements in the devmaps. Emulated device addresses (device numbers) are independent between the instances, and both might use the same addresses.

Each emulated OSA requires its own Ethernet adapter, and two adapters are necessary in this case. Two emulated OSAs cannot share an Ethernet adapter. This example uses LAN Channel Station (LCS) (non-QDIO) mode for both instances, but they both can be QDIO or a mixture of LCS and QDIO.[1] The following devmaps create a tunnel interface for only one of the instances to illustrate that different configurations are possible for independent instances.

Simplified devmaps, matching Figure 10-2 on page 217, might be as follows:

```
(file /home/ibmsys1/aprof1)
      [system]
      memory 6000m                    # emulated zSeries to have 6000 MB memory
      3270port 3270                   # tn3270e connections specify this port.
      processors 2 cp cp

      [manager]
      name awsckd 0001                # Define a single 3390 disk.
      device 0a80 3390 3990 /z/SARES1

      [manager]
      name aws3274 0003               # Define two local 3270 devices.
      device 0700 3279 3274 mstcon
      device 0701 3279 3274 tso

      [manager]
      name awsosa 00C0 --path=F0 --pathtype=OSE
      device E20 osa osa --unitadd=0
      device E21 osa osa --unitadd=1

      [manager]
      name awsosa 00A0 --path A0 --pathtype=OSE --tunnel_intf=y
      device E22 osa osa --unitadd=0
      device E23 osa osa --unitadd=1


(file /home/ibmsys2/profSB)
      [system]
      memory 4000m                    # emulated zSeries to have 4 GB memory
      3270port 3271                   # tn3270e connections specify this port.
      processors 1

      [manager]
      name awsckd 0001                # Define a single 3390 disk.
      device 0a80 3390 3990 /z/SA9999
      device 0200 3390 3990 /z/VMBASE

      [manager]
      name aws3274 0003               # Define two local 3270 devices
      device 0700 3279 3274 L700
```

[1] Because each instance has its own OSA, the user does not need to do OAT configuration if the user uses the default OAT definitions.

```
      device 0701 3279 3274 L701

      [manager]
      name awsosa 0123 --path=F1 --pathtype=OSE
      device E20 osa osa --unitadd=0
      device E21 osa osa --unitadd=1
```

Starting these instances when working from the Linux desktop might go as follows. Log in as root and open a terminal window.

```
# xhost +                               #Allow multiple users to start x3270.
# su ibmsys1
$ cd /home/ibmsys1
$ awsstart aprof1                             #Working as ibmsys1
   (startup messages)                         #ibmsys1 instance
$ x3270 -port 3270 mstcon:localhost &         #Working as ibmsys1
$ x3270 -port 3270 tso:localhost &            #Working as ibmsys1
$ ipl a80 parm 0a8200                         #Working as ibmsys1. Perform an IPL
                                              #of z/OS.
   (open another terminal window)
# su ibmsys2
$ cd /home/ibmsys2
$ awsstart profSB                             #Working as ibmsys2
   (startup messages)                         #ibmsys2 instance
$ x3270 -port 3271 localhost &                #Working as ibmsys2
$ x3270 -port 3271 localhost &                #Working as ibmsys2
$ ipl 200                                     #Working as ibmsys2 Perform an IPL
                                              #of VM.
```

Each instance starts with its own devmap. Each devmap must specify a different port address for local 3270 connections. Each instance must specify different emulated disk volumes. Attempting to share an emulated disk volume in this situation (by specifying the same Linux file for the emulated volume) might result in corrupted data on the volume.

The usage of **xhost +** presents a security exposure. Tailor this command to suit your security environment.

10.4 Instances with shared I/O

It is possible for multiple instances to share certain devices, such as emulated direct access storage device and emulated OSA. Also, a single pool of 3270 devices can be used and accessed through a common Linux port number, although this option has more complex side effects. The most common usage of a shared configuration is to provide *shared direct access storage device* among the instances.

Tip: We assume that readers are familiar with shared direct access storage device. zPDT allows shared emulated direct access storage device, which is equivalent to shared direct access storage device hardware among multiple IBM zSystems servers (or among logical partitions (LPARs)). Shared direct access storage device among multiple z/OS systems typically requires more sophisticated software control, usually through global resource serialization (GRS) and JCL (or equivalent) parameters.

The `--shared` option in a devmap causes zPDT to emulate the `RESERVE` function. zPDT does not automatically emulate GRS or any other software elements that are used to protect user data on shared direct access storage device. This level of protection might be obtained through a Parallel Sysplex operation or through a GRS operation through channel-to-channel (CTC).

ISV zPDT does not support the virtual MAC (VMAC) function of z/OS. The only virtual MAC that is supported is generated on z/VM with the layer-2 vswitch.

A configuration with shared I/O devices requires a *group controller*, as shown in Figure 10-3. The group controller is like another ISV zPDT instance, but without an associated CP or defined memory. The group controller must have its own Linux user ID and devmap, and it starts with its own `awsstart` command. It must be started before other instances are started. As a basic concept, the I/O devices that are defined in the group controller's devmap are inherited and shared by the other instances.

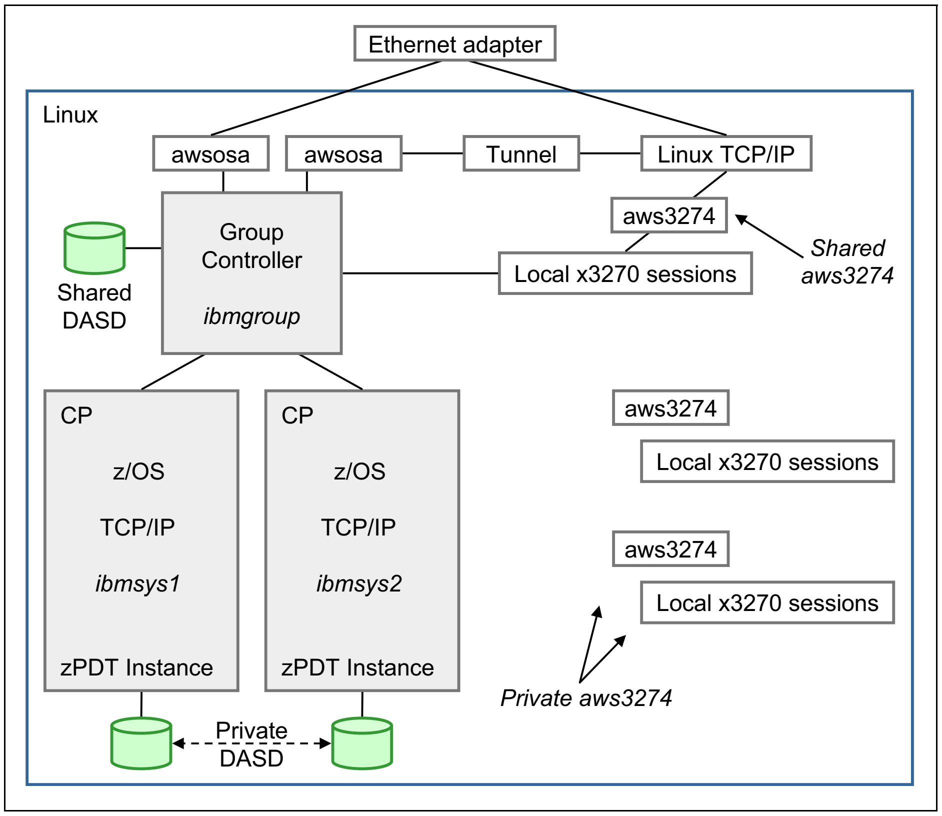


Figure 10-3 Shared emulated I/O

The Linux user ID that is associated with the group controller must have the correct path information that is set in the `.bashrc` file, like that for the user IDs that are associated with each instance. All the user IDs that are involved (the group controller and the instances) must be in the same Linux group, which is group `ibmsys` in our examples.

The Linux user ID (and group ID) for running ISV zPDT is arbitrary. However, there is a special case if you plan to run multiple ISV zPDT instances with a group controller. In this case (with a group controller), the Linux group names must not be the same as the user IDs. For example, if user IDs `ibmsys1`, `ibmsys2`, and `ibmsys3` are used for ISV zPDT with the controller instance (`ibmsys1`) and the ISV zPDT operational instances (`ibmsys2` and `ibmsys3`), then there must not be Linux groups that are named `ibmsys1`, `ibmsys2`, or `ibmsys3`.

With recent Linux distributions, creating a user ID (`ibmsys1`, for example) automatically creates a group with the same name. This action prevents starting an ISV zPDT operation that involves a group controller and one or more ISV zPDT operational instances. The error message is `AWSSTA020E Unable to load DEVMAP file`, which might not be helpful in this situation.

Furthermore, the home directories of the user IDs that are involved (`ibmsys1`, `ibmsys2`, and `ibmsys3` in our example) should be mutually readable/writable. Recent Linux distributions have default home directory permissions of 700 (`rwx------`), which prevents the operational instances from starting under the group controller.[2]

All of the emulated volume files must be readable and writable (possibly through the group ID) by all the user IDs that are involved.[3] For our example, assume that we have three user IDs that are defined (`ibmgroup`, `ibmsys1`, and `ibmsys2`) and all of them are in group `ibmsys`. We can define three devmaps, as follows:

```
/home/ibmgroup/group1
   [system]
   members ibmsys1 ibmsys2        # user IDs for the instances

   [manager]
   name awsckd 8765 --shared
   device A80 3390 3990 /z/Z9RES1
   device A81 3390 3990 /z/Z9RES2
   device A82 3390 3990 /z/Z9SYS1
   device A83 3390 3990 /z/Z9RES3
   device A84 3390 3990 /z/Z9USS1
   device A90 3390 3990 /z/SARES1

   [manager]
   name awsosa 1223 path=F0 --pathtype=OSD
   device 400 osa osa
   device 401 osa osa
   device 402 osa osa
   device 403 osa osa
   device 404 osa osa
   device 405 osa osa
   device 406 osa osa
   device 407 osa osa
```

[2] There are many options for solving this situation, depending on your security requirements. You might, for example, create a unique Linux group for ISV zPDT and allow access to the home directories through that group. The access that is needed is to the z1090 subdirectory in each home directory, and you can arrange suitable permissions to allow only this access.

[3] This setup is controlled by normal Linux permission settings for each file. For example, the command **`chmod g+w /z/*`** can be used to make all the files in directory `/z` writable by members of the current group.

```
/home/ibmsys1/aprof1
   [system]
   memory 8000m
   3270port 3270
   processors 1
   group ibmgroup                    #user ID of the group controller

   [manager]
   name aws3274 4455
   device 0700 3279 3274 mstcon
   device 0701 3279 3274

/home/ibmsys2/aprofSB
   [system]
   memory 5000m
   3270port 3271
   processors 2
   group ibmgroup                    #user ID of the group controller

   [manager]
   name aws3274 5544
   device 0700 3279 3274 mstcon
   device 0701 3279 3274
```

Notice the two new devmap statements in this example. Both are in the `[system]` stanzas:

- **`members name1 name2`** is used in the group controller definitions and specifies the Linux user ID that is associated with each instance in the group.
- **`group cntlname`** is used in each instance and specifies the Linux user ID that is associated with the group controller.

TN3270e sessions are directed to the wanted instance by using the appropriate 3270port number:

```
$ x3270 -port 3270 localhost &            #Connects to the ibmsys1 instance.
$ x3270 -port 3271 localhost &            #Connects to the ibmsys2 instance.
```

There is no need to coordinate device numbers or unit addresses among multiple instances that use shared OSA. For example, each instance might use an OSA interface at addresses 400 - 403. Each instance might start unit addresses (as specified in the devmap) at address zero. (Multiple guests under z/VM, in a single instance, must manage the addresses. Do not confuse multiple guests under z/VM with multiple instances.)

Only direct access storage device (CKD or Fixed Block Architecture (FBA)), aws3274, and OSA can be shared. Additional devices, such as tape drives, can be included in the group controller devmap. These additional device definitions are inherited by all instances, but each instance uses the definitions as though they were part of the devmap for that instance. The two instances in the previous example have different 3270port addresses. We decided to not use shared 3270 definitions in this example.[4] No direct access storage device is defined for the instances in this example, so the instances share the direct access storage device that is defined for the group controller.

[4] An example that uses a single 3270port number is given later in the text.

All sharing instances use the same addresses (device numbers) for the shared devices. There is no provision for different addresses (for different instances) for the same shared device.

If an ISV zPDT instance operates under the group controller, then any OSA devices might be shared devices that are managed by the group controller, or each instance can have a private OSA. If the OSA is used in OSE (non-QDIO mode), then the OAT definitions must be customized with the names of the instance members (specified as "MEMBER names") and the IP addresses for each instance. (As a best practice, use OSD mode.)

Standard operating rules apply. You cannot perform an initial program load (IPL) for the same z/OS system into two instances at the same time.[5] In our small example, we have two z/OS systems (the second one is on the SARES1 volume that is provided with the Application Development Controlled Distribution (ADCD) package). In the absence of shared ENQ functions,[6] you must manage any active data set sharing. The ISV zPDT system correctly emulates disk **RESERVE** and **RELEASE** functions, and they protect the Volume Table of Contents (VTOC), catalog, and some other updates in the normal z/OS manner.

Starting the controller and two instances, working from the Linux desktop, for our example, might go as follows. Log in as root, and open a terminal window:

```
# xhost +                               #Allow multiple users to start x3270.
# su ibmgroup                                 #Work as a group controller.
$ cd /home/ibmgroup
$ awsstart group1                             #Start as a group controller.
   (startup messages)
(open another terminal window)
# su ibmsys1
$ cd /home/ibmsys1
$ awsstart aprof1                             #Working as ibmsys1
   (startup messages)                         #ibmsys1 instance
$ x3270 -port 3270 mstcon@localhost &         #Working as ibmsys1
$ x3270 -port 3270 tso@localhost &            #Working as ibmsys1
$ ipl a80 parm 0a8200                         #Working as ibmsys1. Perform an IPL
                                              #of z/OS.
```

Open another terminal window:

```
# su ibmsys2
$ cd /home/ibmsys2
$ awsstart profSB                             #Working as ibmsys2
   (startup messages)                         #ibmsys2 instance
$ x3270 -port 3271 mstcon@localhost &         #Working as ibmsys2
$ x3270 -port 3271 tso@localhost &            #Working as ibmsys2
$ ipl a90 parm 0a90sa                         #Working as ibmsys2. Perform an IPL
                                              # of z/OS.
```

In this example, we used a different terminal window to start the group controller and each z/OS instance. We can send commands (such as **awsstop**) to the appropriate application later. Three base Linux user IDs are used: `ibmgroup`, `ibmsys1`, and `ibmsys2`.

[5] This statement ignores situations where the usage of different parmlib members allows an IPL of the same z/OS in multiple LPARs or instances. This task involves separate paging, spooling, and various Virtual Storage Access Method (VSAM) data sets for each LPAR or instance. The z/OS ADCD system that is used for many of our examples is not configured for this type of usage.

[6] Sharing ENQ/DEQ functions is typically done by the GRS functions of a sysplex configuration. We do not have a sysplex here, and there are no global ENQ/DEQ controls.

10.5 Additional shared functions

The previous section outlined the key shared device usage rules as they apply to direct access storage device and OSA devices. The group controller also can include a shared aws3270 function and passive definitions that are inherited by all instances.

Shared aws3270 options

The group controller devmap can include aws3270 definitions, such as in this example:

```
/home/ibmgroup/group1
   [system]
   3270port 3270
   members ibmsys1 ibmsys2

   [manager]
   name aws3270 1234
   device 700 3279 3274 mstcon
   device 701 3279 3274
   device 702 3270 3274
```

In this case, the group controller specified a port address for TN3270e connections. Each instance inherits the complete set of 3270 device definitions (700, 701, and 702) but not the 3270port address. Each instance has a 3270 at address 700, 701, and so forth. If a user starts a TN3270e session that is connected to the 3270port number on Linux, the user has several options:

```
$ x3270 -port 3270 localhost &                  #Example 1
$ x3270 -port 3270 ibmsys1@localhost &          #Example 2
$ x3270 -port 3270 ibmsys2.701@localhost &      #Example 3
$ x3270 -port 3270 ibmsys1.mstcon@localhost &   #Example 4
```

In Example 1, the instance is not specified. In this case, the group controller displays a selection menu (on the new 3270 session) and you must indicate which instance you want and, optionally, which terminal in that instance.[7] This selection menu is illustrated in Figure 10-4.

```
                  *** Welcome to the zPDT selection menu ***
               Select the member to connect from the list below
              or type in the member or LU name and depress ENTER

Selection => __    (0 to disconnect)        MEMBER:________  LU:________
1) IBMSYS1
2) IBMSYS2
```

Figure 10-4 Selection menu with two instances running

In Example 2, the first available 3270 in the `ibmsys1` instance is assigned. (The *instance name* corresponds to the Linux user ID that started the instance.) Examples 3 and 4 specify both an instance name and the 3270 device identifier.

[7] If a specific terminal within the instance is not specified (by address or by LU name), then the first available 3270 in that instance is used.

You can specify a different 3270port number and aws3274 device definitions in each instance. In this case, the shared aws3274 conditions do not apply. You can specify a 3270port number and aws3274 devices in the group controller and also specify aws3274 devices in each instance (but without a 3270port number in the instances). In this case, all the 3270 devices (from the controller list that is inherited by all instances, and from the unique list in each instance) can be accessed from the selection menu.

Yet another option exists for accessing shared aws3274 functions. It involves an InetD service that automatically detects which 1090 instances are running and constructs a selection menu that is based on this information. The InetD setup varies with different Linux distributions.

Inherited devices

The group controller devmap can include definitions for device managers other than awsckd, awsfba, awsosa, and aws3270, as in the following example:

```
/home/ibmgroup/group1
[system]
membersibmsys1 ibmsys2

[manager]
name awstape 4444
device 580 3480 3480
device 581 3480 3480
```

In this case, each instance (`ibmsys1` and `ibmsys2`) has emulated 3480 tape drives at addresses 580 and 581. There is no connection between these drives in the two instances. It is exactly as though the awstape stanza appeared in the devmaps for each instance. The sole purpose is to remove the necessity for defining these devices for each instance, which is not meaningful for a small device list as shown here, but might be more meaningful for longer lists.

11

The awscmd device manager

The awscmd device manager provides a “device” that appears to IBM zSystems software as a tape drive. Its function is to send a command (and possibly data) to the underlying Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) host Linux and then receive the output from the Linux command. Any Linux command might be sent, including one that might destroy the Linux system. Obviously, this device manager should be used with care, and it might not be appropriate for an ISV zPDT environment that can be accessed by untrusted users.

The device map (devmap) configuration is like other device managers:

```
[manager]
name awscmd 20
device 560 3480 3480
```

The CUNUMBR (which is 20 in this example) is an arbitrary hexadecimal number (up to 4 hex digits) that cannot duplicate the CUNUMBR that is used with any other device manager. Typically, only one device is used. The device type can be 3420, 3422, 3480, 3490, or 3590, which are the tape device types that are emulated by ISV zPDT. The device number (560 in the example) must match a corresponding device type in your z/OS input/output definition file (IODF). (Any device number can be used with z/VM.)

The intended operation is as follows:

1. A rewind is issued to the device.
2. The Linux command (expressed in Extended Binary Coded Decimal Interchange Code (EBCDIC)) is written to the device.
3. Any stdin data that is used by the Linux command is written to the device.
4. EBCDIC to ASCII translation is done automatically. Binary data is not possible.
5. A tape mark is written to the device.
6. The awscmd device manager submits the command (and data) to Linux through a shell that does not appear in a Linux window. The Linux current directory for the command is the directory that was used to start ISV 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) is converted to EBCDIC.
9. Two tape marks are at the end of the pseudo-tape.

Restrictions

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

At the time of writing, the following characters did not survive the conversion from EBCDIC to ASCII when included in SYSIN data:

- Tilde (~)
- Caret (^)
- Colon (:)
- Double quotation marks (")
- Less than (<)
- Greater than (>)
- Question mark (?)

11.1 Sample z/VM script

The following REXX script assumes that the awscmd device is attached to the Conversational Monitor System (CMS) user as device 28F:

```
/* CMS REXX script to run a Linux command    */
/*                                           */
/* format:                                   */
/*    oscmd Linux-command (tape-address      */
/*                                           */
/* The tape-address is optional; defaults to 28F */
/*                                           */
 Trace off;
 Parse arg cmd '(' tDev;
 if (length(tDev) = 0)
  then tDev = 28F;
/* Write the Linux command string to the tape    */
 “tape rew (“ tDev;
 “pipe var cmd | tape” tDev;
 “tape wtm (“ eDev;
/* Read the stdout file from the tape           */
 “tape rew (“ tDev;
 “tape fsf (“ tDev;       /* skip over input file */
 say “STDOUT output------”
 “pipe tape” tDev “| console”;
```

```
/* Read the stderr file from the tape           */
 "say "STDERR output ------"
 "pipe tape" tDev "| console"
/* Read the return code from the tape           */
 "pipe tape" tDev "| console"
/* end this script                              */
 return(0);
```

This script can be used from z/VM as follows:

```
att 280 * 28f           #(Attach the awscmd pseudo-device.)
TAPE 0280 ATTACHED TO ZVMTEST 028F
Ready;
oscmd ls -al
STDOUT output-------
total 21699469
drwxrwxr-x 2 zvmtest zvmtest       4096 Aug  7 20:51 .
drwdr-xr-x 8 zvmtest zvmtest       4096 Aug  7 20:37 ..
-rw-rw-r-- 1 zvmtest zvmtest 2846431232 Jul  8 09:58 530OPT.ckd
-rw-rw-r-- 1 zvmtest zvmtest 2846431232 Jul  8 10:08 530PAG.ckd
   #(etc to list all the entries in the current Linux directory)
STDERR output----
COMMAND return code ----
0
Ready;
```

11.2 z/OS use

Using the awscmd device with z/OS is more challenging than using it with z/VM for several reasons:

- Tape drives are not readily manipulated by Time Sharing Option (TSO) users.
- z/OS wants to check tape volumes for labels, even if you specify a no-label tape volume.
- For practical purposes, an assembly program must be written to use the **awscmd** functions.

You can write your own program to use the awscmd function. You might want to examine the sample program in 11.2.1, "Sample z/OS program for awscmd" on page 231. This program looks for a PARM field on the **EXEC JCL** statement and sends this field as the command to Linux. If no PARM field is present, it opens **DDname SYSIN** and sends the first data line as the Linux command, and sends any additional data lines as stdin for the Linux command. Output from the awscmd is printed on **DDNAME SYSPRINT**. A **JCL DD** statement is needed to allocate the pseudo-tape drive for awscmd. Our example uses address 560 for the pseudo-tape because it is a known 3480 address for our z/OS system.

Our devmap contains these lines:

```
[manager]
name awscmd 20
device 560 3480 3480
```

Our sample program requires that an MVS initiator is enabled for Bypass Label Processing (BLP)[1] processing. You can accomplish this task by changing the Job Entry Subsystem (JES) 2 startup parameters or (during an initial program load (IPL)) by entering the following command on the MVS console:

```
$T JOBCLASS(A),BLP=YES           (You might want to use a jobclass other than A.)
```

Then, mount a tape on the pseudo-tape drive for continued usage so that you do not need to respond to a mount message every time that the sample program runs:

```
MOUNT 560,VOL=(NL,123456)
```

The MVS **mount** command followed by the ISV zPDT **ready** command provides the necessary setup:

```
$ ready 560
```

An example of using the sample program might be as follows:

```
//OGDEN22 JOB 1,OGDEN,MSGCLASS=X,MSGLEVEL=(0,0)
//  EXEC PGM=AWSCMDX,PARM='ls -al '
//STEPLIB DD DSN=OGDEN.LIB.LOAD,DISP=SHR
//TAPE DD UNIT=(560,,DEFER),VOL=SER=123456,LABEL=(1,BLP),DSN=X
//SYSPRINT DD SYSOUT=*
```

The **TAPE DD** statement is used internally by awscmd, so nothing is written to the device. The job output (viewed from the JES2 spool by using System Display and Search Facility (SDSF)) contains the usual JES2 messages and a SYSOUT data set, such as the following example:

```
COMMAND:  ls -al 1>/tmp/AWSCMD-xxx-out.txt 2>/tmp/etc.xxetc  </tmp/AWSCMD.xxxetc
STDOUT: total 21699469
STDOUT: drwxrwxr-x 2 ibmsys1 ibmsys1       4096 Aug  1 20:01 .
STDOUT: drwdr-xr-x 4 ibmsys1 ibmsys1       4096 Aug  1 20:02 ..
STDOUT: -rw-rw-r-- 1 ibmsys1 ibmsys1 2846431232 Aug  8 09:58 WORK01
STDOUT: -rw-rw-r-- 1 ibmsys1 ibmsys1 2846431232 Aug  8 10:08 WORK02
   (etc to list all the entries in the current Linux directory)
STDERR:
RTNCDE: 0
```

The Linux command that is run contains redirection operators that are automatically added by awscmd. You can see these operators in the output listing. They are only suggested in the sample output that is shown here.

A second example, which uses SYSIN data to create a Linux file, might be as follows:

```
//OGDEN22 JOB 1,OGDEN,MSGCLASS=X,MSGLEVEL=(0,0)
//  EXEC PGM=AWSCMDX
//STEPLIB DD DSN=OGDEN.LIB.LOAD,DISP=SHR
//TAPE DD UNIT=(560,,DEFER),VOL=SER=123456,LABEL=(1,BLP),DSN=X
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
tee my.new.file
This is a line of data for the new file
This is another line of data
And yet another
/*
```

[1] BLP prevents z/OS from testing for a standard label on the tape.

This example creates the Linux file `my.new.file` (in the Linux directory that is used to start ISV zPDT) with the indicated lines in the file. A fully qualified Linux file name can be used with either of the examples. The output for the second example lists all the data lines with the **COMMAND** output, and then lists them again for **STDOUT** output because **tee** writes all the stdin data to stdout. The **tee** command appends the new data if the specified file exists.

11.2.1 Sample z/OS program for awscmd

The following listing is a z/OS batch program that is called AWSCMD, which uses the awscmd command processor to send a command to the underlying Linux and receive the results. This sample program is simple-minded in many respects and not intended to illustrate the best programming techniques, but it should be fairly readable. Errors result in Write To Operator (WTO) messages, which is a poor design for significant applications, but it should be reasonable for many ISV zPDT environments. The program includes a 2-second wait before reading the results of the Linux command. This wait is not necessary and can be removed. (The pseudo-tape device remains busy with the preceding Write Tape Mark (WTM) operation until the Linux command completes.)

The following listing contains a minor update from previous versions because it uses a TRTR instruction to “backscan” for blank characters:

```
//OGDEN90  JOB 1,OGDEN,MSGCLASS=X
//  EXEC  ASMACLG,PARM.C='NOXREF',PARM.L='NOLIST,NOMAP'
//C.SYSIN  DD *
*
* Send a command to Linux through AWSCMD and read the result.
* JES2 must allow BLP for the job class that is used.
*       $T JOBCLASS(A),BLP=YES can be used for temporary changes.
*
* Need to mount "dummy" tape: MOUNT 560,VOL=(NL,123456)
*
*//TAPE  DD UNIT=(560,,DEFER),LABEL=(1,BLP),VOL=SER=XXXXXX,DSN=X
*//SYSPRINT DD SYSOUT=*     (output from Linux)
*   (You can omit printing of the "returned" files.)
         PRINT NOGEN
AWSCMD   CSECT
         STM   14,12,12(13)    Save caller's registers
         LR    12,15           Use entry-point as base register
         USING AWSCMD,12
         ST    13,SAVEAREA+4   Caller's SAVEAREA address
         LA    2,SAVEAREA      My SAVEAREA address
         ST    2,8(13)         Store @(MY SAVE AREA) in caller
         LR    13,2            My SAVEAREA in R13     )
         LR    11,12           Second base register if needed
         A     11,=F'4096'
         USING AWSCMD+4096,11
* Check PARM data
         L     1,0(1)          Get address of PARM field
         LH    2,0(1)          Get length of PARM field (if any)
         LTR   2,2             Check it
         BZ    NOPARM          Branch if length = zero (no PARM)
         BCTR  2,0             Subtract 1 from length (for MVC)
         EX    2,MOVPARM       Move PARM to your work area
         B     A               Skip SYSIN usage
* If no PARM, use SYSIN for command and data
```

```
NOPARM   OPEN  (SYSIN,(INPUT))
         TM    SYSIN+48,X'10'  Did OPEN work?
         BZ    NOINPUT         If not, branch
         MVI   PARMFLG,X'FF'   Remember to use SYSIN data
* Open psudo0-tape device and SYSPRINT
A        OPEN  STAPEO          It should have BLP on statement
         TM    STAPEO+48,X'10' Did OPEN work?
         BZ    ERR10           If not, branch
A10      OPEN  (PRINT,(OUTPUT))
         TM    PRINT+48,X'10'  Did OPEN work?
         BZ    ERR11
* Rewind the pseudo tape (might not be necessary but does not hurt)
B        MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCREW        address of CCW
         ST    1,SIOB+16       Place @(CCW) in IOB
         EXCP  SIOB            Run the rewind CCW
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      Completed OK?
         BZ    ERR1            If not, branch
* Prepare to write the Linux command (and any additional data)
C        CLI   PARMFLG,X'00'   Using PARM data?
         BE    C2              If so, branch
C1       GET   SYSIN,BUFFER    If not PARM, then read SYSIN
* Backscan to remove trailing blanks
C2       TRTR  BUFFER+99(100),TRTRFUNC  Backscan for first non-blank
         BZ    ERR6            Branch if no non-blank was found
         LA    3,BUFFER        @(Beginning of buffer)
         SR    1,3             Subtract from @(non-blank character)
         LA    1,1(1)          Add 1 to get length
*C5       CLC   SCCWRITE+6(2),=X'0000'  Is the CCW length already set?
*         BNE   C6              If so, branch .............
         STH   1,SCCWRITE+6    Set the CCW length field
C6       MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCWRITE      Get @(write CCW)
         ST    1,SIOB+16       Put it in the IOB
         EXCP  SIOB            Run the write
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BZ    ERR2            If not OK, branch
         CLI   PARMFLG,X'00'   PARM data?
         BE    D               If so, there is no additional data
         MVC   BUFFER(132),BLANKS
         B     C1              If SYSIN, loop to read data records
* Write a tape mark to tell awscmd to send the material to Linux
D        MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCWTM        Address of WTM CCW
         ST    1,SIOB+16       Put it in the IOB
         EXCP  SIOB            Write tape mark
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BZ    ERR3            If not, branch
         B     E
*
* WAIT AN ARBITRARY TWO SECONDS FOR LINUX
*
```

```
*E       STIMER WAIT,BINTVL=SEC2
*
* Rewind the pseudo tape
E        MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCREW        Address of rewind CCW
         ST    1,SIOB+16       Put address in the IOB
         EXCP  SIOB            Rewind the pseudo tape
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BZ    ERR4            If not, branch
* Read the first file that awscmd put on the pseudo tape
F        MVC   BUFFER2(80),BLANKS    Allow for a fairly large record
         MVC   BUFFER2+0080(80),BLANKS
         MVC   BUFFER2+0160(80),BLANKS
         MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCREAD       Address of a READ CCW
         ST    1,SIOB+16       Put address in the IOB
         EXCP  SIOB            Do the READ
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BO    F1              If yes, branch
         TM    SIOB+12,X'01'   Unit exception means a tape mark
         BO    G               If tape mark, branch
         B     ERR5            Must be some other problem
F1       MVC   BUFFER(132),BLANKS
         MVC   BUFFER(09),=C'COMMAND: '
         MVC   BUFFER+09(132),BUFFER2
         PUT   PRINT,BUFFER
         B     F               Loop until first file is read
* Read the second file that awscmd put on the pseudo tape
G        MVC   BUFFER2(80),BLANKS
         MVC   BUFFER2+0080(80),BLANKS
         MVC   BUFFER2+0160(80),BLANKS
         MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCREAD       Address of a READ CCW
         ST    1,SIOB+16       Put address in IOB
         EXCP  SIOB            READ
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BO    G1              If you, branch
         TM    SIOB+12,X'01'   Was it a tape mark?
         BO    H               If a TM, branch
         B     ERR5            Some other error
G1       MVC   BUFFER(132),BLANKS
         MVC   BUFFER(09),=C'STDOUT:  '
         MVC   BUFFER+09(132),BUFFER2
         PUT   PRINT,BUFFER
         B     G               Loop until all records in file are read
* Read third file that is returned from Linux
H        MVC   BUFFER2(80),BLANKS
         MVC   BUFFER2+0080(80),BLANKS
         MVC   BUFFER2+0160(80),BLANKS
         MVI   SECB,X'00'      Zero the ECB
         LA    1,SCCREAD       Address of CCW
         ST    1,SIOB+16       Put address in IOB
         EXCP  SIOB            READ
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BO    H1              If yes, branch
         TM    SIOB+12,X'01'   Was it a tape mark?
         BO    J               If yes, branch
         B     ERR5            Branch if some other error
H1       MVC   BUFFER(132),BLANKS
         MVC   BUFFER(09),=C'STDERR:  '
         MVC   BUFFER+09(132),BUFFER2
         PUT   PRINT,BUFFER
         B     H               Loop until all records in file are read
* Read fourth file that is returned from Linux
J        MVC   BUFFER2(80),BLANKS ux
         MVC   BUFFER2+0080(80),BLANKS
         MVC   BUFFER2+0160(80),BLANKS
         MVI   SECB,X'00'      Zero your ECB
         LA    1,SCCREAD       Address of CCW
         ST    1,SIOB+16       Put address in IOB
         EXCP  SIOB            READ
         WAIT  ECB=SECB        Wait for it
         TM    SECB,X'20'      OK?
         BO    J1              If OK, branch
         TM    SIOB+12,X'01'   Got a tape mark?
         BO    K               If yes, branch
         B     ERR5            Some other error
J1       MVC   BUFFER(132),BLANKS
         MVC   BUFFER(09),=C'RTNCDE:  '
         MVC   BUFFER+09(132),BUFFER2
         PUT   PRINT,BUFFER
         B     J               Loop until all records in file are read
*
K        SR    1,1             NOP
*
CLOSE    CLOSE (STAPEO,,PRINT)
RETURN   L     13,4(13)        Get @(caller's save area)
         LM    14,12,12(13)    Restore caller's registers
         SR    15,15           Set return code
         BR    14              Exit
*
*   Errors and messages
*
ERR1     MVC   BUFFER(132),BLANKS
         MVC   BUFFER(21),=C'Initial rewind failed'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR2     MVC   BUFFER(132),BLANKS
         MVC   BUFFER(12),=C'Write failed'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR3     MVC   BUFFER(132),BLANKS
         MVC   BUFFER(22),=C'Write tape mark failed'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR4     MVC   BUFFER(132),BLANKS
```

```
         MVC   BUFFER(20),=C'Second rewind failed'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR5     MVC   BUFFER(132),BLANKS
         MVC   BUFFER(17),=C'First read failed'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR6     MVC   BUFFER(132),BLANKS
         MVC   BUFFER(17),=C'No non-blank character found'
         PUT   PRINT,BUFFER
         B     CLOSE
ERR10    WTO   'NO TAPE DD STATEMENT'
         B     RETURN
ERR11    WTO   'NO SYSPRINT DD STATEMENT'
         B     CLOSE
NOINPUT  WTO   'NO PARAMETER OR SYSIN FOUND'
         B     RETURN
*
*  Constants, work areas
*
SAVEAREA DC    18F'0'
SEC2     DC    F'200'          TWO SECONDS
STAPEO   DCB   DSORG=PS,MACRF=(E),DDNAME=TAPE
PRINT    DCB   DSORG=PS,DDNAME=SYSPRINT,MACRF=(PM),LRECL=132,          X
               BLKSIZE=13200,RECFM=FB
SYSIN    DCB   DSORG=PS,DDNAME=SYSIN,MACRF=(GM),LRECL=80,              X
               RECFM=FB,EODAD=D
         DS    D'0'
SCCWRITE DC    X'01',AL3(BUFFER),X'20',AL3(0)
SCCWTM   DC    X'1F',AL3(0),X'20',AL3(1)
SCCREW   DC    X'07',AL3(BLANKS),X'20',AL3(1)
SCCREAD  DC    X'02',AL3(BUFFER2),X'20',AL3(3200)
SECB     DC    F'0'
SIOB     DC    X'42000000'
         DC    A(SECB)          A(ECB)
         DC    2F'0' CSW
         DC    A(0)             A(CCW)
         DC    A(STAPEO)        A(DCB)
         DC    2F'0'
*
PARM     DC    CL100' '        PARM can be up to 100 characters
PARMFLG  DC    X'00'           00=PARM, FF=SYSIN
MOVPARM  MVC   BUFFER(0),2(1)  Used by EX instruction
BLANKS   DC    CL132' '
BUFFER   DC    CL132' '
TRTRFUNC DC    64X'01'         Function table for TRTR instruction
         DC    X'00'
         DC    191X'01'
         LTORG
BUFFER2  DS    CL4000' '
         END
/*
//*.SYSLMOD DD DISP=OLD,DSN=OGDEN.LIB.LOAD(AWSCMD)
//G.TAPE  DD UNIT=(560,,DEFER),LABEL=(1,BLP),VOL=SER=123456,DSN=X
//G.SYSPRINT DD SYSOUT=*
```

```
//G.SYSIN DD *
ls -al
/*
```

12

Minor z/OS notes

This chapter is not intended as a general z/OS guide, or as a guide to the Application Development Controlled Distribution (ADCD) systems. However, several common questions or problems are discussed here.

12.1 Minor z/OS tuning or adjustment options

The ADCD z/OS systems also are used on full-sized mainframes. Some of the default options might not suit smaller Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) users. Here are some small, minor optional changes that might be helpful, but these details might change with new z/OS ADCD releases:

- The Health Checker is not needed.

  In IEASYSxx, insert the line `HZSPROC=*NONE,` (remember the comma). (This parameter was fully effective when tried at the time of writing.)

  - To manually start Health Checker, run **`S HZSPROC,HZSPRM='PREV'`**.
  - To manually stop Health Checker, run **`P HZSPROC`**.
  - To display the Health Checker's status, run **`F HZSPROC,DISPLAY,STATUS`**.

- z/OSMF is not needed.

  In IEASYSxx, change `IZU=xx,` to `IZU=NO,` (remember the comma).

  For manual starting or stopping, use the following commands:

  - To start z/OSMF, run the following commands:

    ```
    S CFZCIM
    S IZUANG1
    S IZUSVR1
    ```

  - To stop z/OSMF, run the following commands:

    ```
    P IZUSVR1
    P IZUANG1
    P CFZCIM
    ```

- To prevent SMF logging, change the SMFPRM00 first line from `ACTIVE` to `NOACTIVE`.
- SYS1.LOGREC is full.

  Maintaining SYS1.LOGREC[1] is a normal z/OS system programmer's task. There is nothing unique to ISV zPDT or the ADCD z/OS distribution. Production installations, which use larger IBM zSystems machines, often keep ordered histories of LOGREC data and study any new material in LOGREC. The histories include data about hardware and software failures, initial program load (IPL) statistics, volume use statistics, and so forth.

  Most ISV zPDT users ignore SYS1.LOGREC until they receive messages that it is full. It is a best practice to clear LOGREC when such messages are received. There is at least one job in the ADCD libraries to do this task, but the job name (and library name) might change from time to time.

  Here is a job to clear SYS1.LOGREC. The particular format that is shown here is old, but it can be used exactly as shown.

  ```
  //BILLCL  JOB  1,OGDEN,MSGCLASS=X
  //  EXEC  PGM=IFCEREP1,PARM='CARD'
  //SERLOG  DD  DISP=SHR,DSN=SYS1.S0W1.LOGREC
  //DIRECTWK  DD  UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
  //EREPPT  DD  SYSOUT=*,DCB=BLKSIZE=133
  //ACCDEV DD DUMMY
  //TOURIST  DD  SYSOUT=*,DCB=BLKSIZE=133
  //SYSIN DD *
  ```

[1] SYS1.LOGREC is the traditional name for this data set; however, installations can change it. Recent z/OS ADCD systems use SYS1.S0W1.LOGREC to allow a logical step to use the name SYS1.S0W2.LOGREC for another z/OS in a sysplex configuration.

```
  SYSUM
  ACC=Y
  ZERO=Y
/*
```

You might find slightly different jobs that perform the same function, and any of them should be acceptable. Never attempt to "clear" SYS1.LOGREC by deleting and reallocating it, or by removing records with a text editor.

You can edit IEASYSxx members (in PARMLIB) and set `LOGREC=IGNORE` to avoid the LOGREC full problems.

- Eliminate the minor scrollable area at the bottom of the operator console.

  Set either of the following strings:

  – `K A,NONE`

  – `AREA(NONE)` in parms for `CONSOL00 DEVNUM(0700)`

- The TCP/IP configuration parameters have become increasingly complex in recent ADCD z/OS releases, which might create problems for some users. The following strings might help resolve the situation in basic cases. (Column 72 continuation indicators are needed for VTAMLST entries but are not shown in the following listings.)

```
ADCD.Z24C.VTAMLST(ATCCON00) <-- data set and member name
   A0600,NSNA70X,DYNMODEL,COSAPPN,
   TCP,OSATRL2,IMS15APL,IMS14APL,CICSAPPL,DBCGLU,DBBGLU

ADCD.Z24C.VTAMLST(OSATRL2) <-- data set and member name
   OSATRL1 VBUILD TYPE=TRL
   OSATRL1E TRLE LNCTL=MPC,READ=(0400),WRITE=(0401),DATAPATH=(0402),
                PORTNAME=PORTA,MPCLEVEL=QDIO
   OSATRL2E TRLE LNCTL=MPC,READ=(0404),WRITE=(0405),DATAPATH=(0406),
                PORTNAME=PORTB,PCLEVEL=QDIO

ADCD.Z24C.TCPPARMS(PROF9) <-- create new member named PROF9
   ARPAGE 5
   DATASETPREFIX TCPIP
   AUTOLOG 5
       FTPD JOBNAME FTPD1   ; FTP Server
       PAGENT               ; Policy Agent Server
       PORTMAP              ; Portmap Server
   ENDAUTOLOG
   PORT
   ..Copy part or all of the assigned port numbers
   ..provided in the ADCD system. You can adjust them as needed
   ..but it might be easiest to copy the whole list
   ;
   SACONFIG DISABLED
   ;
   ;This device defines the tunnel
   DEVICE PORTA  MPCIPA
   LINK ETH1  IPAQENET PORTA
   HOME 10.1.1.2 ETH1
   ;
   ; This second device is optional
   DEVICE PORTB    MPCIPA
   LINK ETH2 IPAQENET  PORTB
   HOME 192.168.1.61   ETH2        <--select an appropriate IP address
```

```
;
BEGINRoutes
;     Destination   SubnetMask    FirstHop       LinkName  Size
ROUTE 192.168.1.0 255.255.255.0       =        ETH2 MTU 1492
ROUTE 10.0.0.0    255.0.0.0           =        ETH1 MTU 1492
ROUTE DEFAULT                     192.168.1.1  ETH2 MTU 1492
ENDRoutes
;
ITRACE OFF
IPCONFIG NODATAGRAMFWD
UDPCONFIG RESTRICTLOWPORTS
TCPCONFIG RESTRICTLOWPORTS
TCPCONFIG TTLS
START PORTA
START PORT
```

```
ADCD.Z24C.PROCLIB(TCPIP)
   //PROFILE DD DISP=SHR,DSN=ADCD&SYSVER..TCPPARMS(PROF9) <--new member name
```

There are multiple arbitrary choices in this example, such as the IP addresses and Open Systems Adapter (OSA) addresses, and it is presented as a starting point. In this example, it is necessary to change the **PROFILE DD** statement in the TCP/IP startup procedure to point to member PROF9. The example data sets are for the ADCD z/OS 2.4C release, but these steps can be adapted to other releases.

- Many ADCD z/OS system data sets appear 100% full. How do you apply maintenance?

  Many ISV zPDT z/OS users do not apply maintenance (program temporary fixes (PTFs)) between ADCD z/OS releases. If you want to apply maintenance or changes,[2] you might need to copy the z/OS system data set to a larger allocation, recatalog it, and then apply maintenance. This procedure often involves performing an IPL of z/OS after the copy and catalog steps.

- The /u file system is small.

  Some ADCD releases have a small file system for /u, which is the most common "location" for a typical omvs user to begin working with their own files. A more skilled omvs user can create their own ZFS file system[3] and mount it in a /u subdirectory. In some cases, /u (under subdirectories, such as /u/ibmuser) contains sizable pax files that might be deleted. There is no single solution for everyone.

- How much storage should there be for z/OS?

  Starting with z/OS 2.3, a minimum of 2 GB of IBM zSystems storage (memory) is required[4] when running z/OS under ISV zPDT. (A minimum of 8 GB is required on other systems.) Your storage requirements depend on what you are running with z/OS. The 2 GB size might be practical for a simple Time Sharing Option (TSO) system with a few users and no major subsystems running. It is not a practical system for most ISV zPDT customers. As always, paging levels in z/OS and the base Linux should be monitored. Most ISV zPDT users have at least 8 GB for z/OS, and some users have more storage.

[2] These comments ignore the principle that normal ISV zPDT licenses do not include simple methods to obtain PTFs or other maintenance software.

[3] This task might be most easily done by using the ish shell "file systems" function.

[4] "Required" might be too strong. A console message is produced if the ISV zPDT system is below the 2 GB size, but the operator can decide to continue the operation.

- Delete log streams.

  The default location for log streams (for the typical ADCD system) is the xxSYS1 volume. These streams can sometimes become too large. A sample job for deleting a log stream is as follows:

  ```
  //BILL1 JOB 1,OGDEN,MSGCLASS=X
  // EXEC PGM=IXCMIAPU
  //SYSPRINT DD SYSOUT=*
  //SYSIN DD *
    DATA TYPE(LOGR) REPORT(YES)
    DELETE LOGSTREAM NAME(WAS.ERROR.LOG)
  /*
  ```

  This task should be done only if the log stream file is not being used or managed of z/OS. (This topic can be complicated for z/OS system managers.)

- Unable to start IBM Interactive System Productivity Facility (ISPF).

  When logging on to TSO, the following message is sometimes seen when attempting to start ISPF:

  ```
  %%% UNABLE TO ALLOCATE OR CREATE ISPF PROFILE DATASET
  ISPF003 FOLLOWING FILE WAS NOT PREALLOCATED
  ISPPROF
  ```

  The most common reason for this problem is that the required ISPF profile data set was uncatalogued for some reason, which can happen when attempting to use the same profile data set from two different z/OS instances, such as in an IBM Parallel Sysplex environment that was not configured for such use. z/OS failing at the wrong moment might also lead to this error.

  You must recatalog the profile data set. Assuming that you are logging on as `IBMUSER`, the data set you want (in all recent ADCD releases) is `IBMUSER.ISPF.ISPPROF`. You can log on with another user ID (`ADCDMST` is convenient for this purpose) and recatalog the data set. The easiest way to recatalog the data set is to list the volume (C4SYS1, for example) by using ISPF 3.4, and then use a **C** command to catalog the data set. (The **C** is entered at the beginning of the line for that data set in the ISPF 3.4 display.)

  Another solution after ISPF fails to start is to preallocate the data set (by using basic TSO commands) and then start ISPF again. To do this task, use the following TSO command:

  ```
  READY alloc da('ibmuser.ispf.ispprof') f(ispprof) shr vol(c4sys1) unit(3390)
  READY ispf                          (Start ISPF again.)
  ```

  You must use the volser that matches your current system.

- LPA and STEPLIBs.

  In the most normal situations when z/OS is looking for an execution program, a STEPLIB or JOBLIB is tried before LPA (or LINKLIB). If the goal is to improve performance for a program that is frequently used, copying it to one of the LPA libraries can be a good move in some cases, such as reentrant programs or program size. If this task is done (a move to an LPA library), ensure that the same program is not found in a STEPLIB or JOBLIB for the frequently run jobs. If the STEPLIB or JOBLIB exists, then the LPA location is ignored and no performance improvements occur.

  For example, frequent usage of the COBOL compiler might be a reason for copying some of the modules to an LPA library; however, the standard catalog procedures in recent ADCD z/OS releases have STEPLIB statements for the compiler.

- IBM zSystems Integrated Information Processor (zIIP) performance.

  Some users noticed that their system can be much slower when a zIIP is configured. Why? A recent article in the Watson & Walker Tuning Newsletter[5] pointed to a cause. There is a special aspect of the workload manager (WLM) service class goal of "discretionary." If a zIIP exists and the workload is zIIP-eligible and in a discretionary service period, the workload is dispatched *only* on zIIP processors even if CPs are idle.

  As an example that involves the current[6] z/OS ADCD systems, if you are running z/OSMF with your ISV zPDT system that has 3 CPs, you see a specific period for z/OSMF start. Then, if you add a zIIP, the z/OSMF start takes much longer because much of z/OSMF is dispatched only to the single zIIP processor. This situation occurs for any application that has large amounts of zIIP-eligible code.

  If you define your own WLM, then you know of any discretionary periods in it. If you are using the recent default z/OS ADCD WLM (or perhaps an unknown WLM), you might investigate it as follows (by using the standard ISPF facilities in the z/OS ADCD system):

  a. Go to option `M` and then option `15 (WLM)`.
  b. Select `Extract definition from WLM couple dataset.`
  c. Select option `4 (Service Classes)`. You should see a list of 10 - 15 service classes. Investigate each one.
  d. Use option `4 (Browse)` for a service class and look at the Goal Description. If it has "discretionary" as the last or only goal, note the service class name. Do not depend on the name as an indication of the purpose of that service class.
  e. Go back to the first WLM menu and select option `6 (Classification Rules)`. Find the service class names that you noted in the "Service" column. Look at the "Type" column, which has meaningful names that indicate the types of workloads that use the indicated service class.

  In the current z/OS ADCD system at the time of writing, service class STCLOM was the only class that had a discretionary period. Workload types LSFM, SOM, and Started Class (STC) use this service class. STC workloads are important for many things, and with these WLM definitions, they can encounter the performance problem that is described here.

- Large SMF data output.

  Some of the recent ADCD z/OS releases produce what some users regard as excessive SMF output, especially if type 99 records are included.

  If you have no interest in SMF data, alter the SMFPRMxx members in PARMLIB and change the first line from `ACTIVE` to `NOACTIVE`. If you might have some interest in SMF job data (type 30 records, for example) and perhaps IBM Resource Measurement Facility (RMF) data (types 70 - 79), add the following line to produce minimal recording:

  ```
  SYS(TYPE(30,70:79),EXITS(IEFU83,IEFU84,IEFACTRT,IEFUSI,IEFUJI,
     IEFU29),NOINTERVAL,NODETAIL)
  ```

  Recent ADCD systems process full SMF data sets automatically through a supplied IEFU29 exit, and then discard the output. If you want to process SMF data yourself, remove the IEFU29 exit that appears twice in the default SMFPRMxx parameters. (As a best practice, eliminate recording SMF record types that you do not want before doing this action.)

[5] This information comes from newsletter that requires a paid subscription to `http://www.watsonwalker.com.` This newsletter is used by many z/OS customers.

[6] The November 2019 and May 2020 z/OS ADCD releases. Future releases might change this configuration.

The uses of the various exits that are specified in the SMF parameters, if present, is beyond the scope of this publication. If you have access to Cheryl Watson Tuning Letters, the 2009 letters contain a useful and practical introduction to SMF usage.[7]

If you use log streams for SMF and want to determine the quantity of each SMF type that is collected, use the following job:

```
//OGDEN101 JOB 1,OGDEN,MSGCLASS=X
//  EXEC PGM=IFASMFDL,REGION=0M
//OUT     DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 LSNAME(IFASMF.GENERAL,OPTIONS(DUMP))
 OUTDD(OUT,TYPE(0:255))
/*
```

This job assumes that your LOGSTREAM for SMF is named `IFASMF.GENERAL`, which is the name that is used in the Parallel Sysplex system for ISV zPDT that is described in other documents. Your name might differ.

If you want to list a few details about the status of the LOGSTREAM, use the following job:

```
//OGDEN102 JOB 1,OGDEN,MSGCLASS=X
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 DATA TYPE(LOGR) REPORT(NO)
 LIST LOGSTREAM NAME(IFASMF.GENERAL) DETAIL(YES)
/*
```

- Excessive Health Checker messages.

  The Health Checker is started automatically under z/OS 2.1 and later. Although it is possible to turn off some of its operations, it is a useful tool. You might find that some of the checks are not relevant to ISV zPDT users or not interesting to you. For example, when running a Parallel Sysplex system,[8] you might see the following message on the MVS console:

```
05.38.33 SOW1 STC00783  HZS0002E CHECK(IBMXCF,XCF_CF_STR_NONVOLATILE):
     IXCH0222E A coupling facility structure user request for non-volatility
     and failure-isolation from connectors is not satisfied.
```

  The message is repeated at intervals. You can delete this example health check as follows:

  a. In an earlier ADCD z/OS 2.4 system,[9] IBMUSER has only *read* access to the Health Checker controls.[10] You need *update* access. Issue the following commands (when logged on as IBMUSER) in the ISPF option 6 panel:

```
PERMIT HZS.** CLASS(XFACILIT) ID(IBMUSER) ACC(CONTROL)
SETROPTS RACLIST(XFACILIT) REFRESH
```

  b. Go the System Display and Search Facility (SDSF) primary option menu and select option CK to access the Health Checker. Scroll through the checks under the *Status* column at the right of the panel. This column contains text such as `SUCCESS`, `INACTI`, and `EXCEPT`. Enter the letter "`S`" at the beginning of any line to see more detail about it. The `EXCEPT` text means that the Health Checker found a problem with this item.

[7] For more information, see http://www.watsonwalker.com.
[8] This Parallel Sysplex system is described in *zPDT Sysplex Extensions - 2020*, SG24-8386.
[9] The details might change with newer ADCD z/OS releases, but the general principles that are explained here might still apply.
[10] User ADCDMST has *update* access to the relevant IBM RACF profile in this particular ADCD release. If this situation is true in your later release, you can log on with this user ID and skip the RACF change.

c. Find the line for the offending check condition and enter the letter “`H`” on the line to make the check inactive or “`P`” to delete the check.

- z/OS spin loop timeouts.

  A z/OS spinloop might time out and produce an S071 ABEND. This event can be triggered by many different circumstances. You can change this time value by creating or altering member EXSPATxx in PARMLIB:

  ```
  member EXSPAT00
     SPINRCVY ABEND
     SPINTIME=60
  ```

  SPINLOOP messages should be rare. The most common reasons for SPINLOOP problems are an overcommitted virtual environment, disk cache writes when a large amount of personal computer (PC) memory is used, or there are high I/O rates from multiple application tasks. In rare cases, a situation with hyper-threading produced SPINLOOP delays. Frequent messages indicate a need to examine your operational configuration carefully.

- PER traps.

  The use of PER traps (typically with the **`SLIP`** command) causes ISV zPDT to stop using its Just In Time (JIT) internal compiler while PER is active. This situation can result in a much slower ISV zPDT operation. As a best practice, avoid PER traps unless you need them.

- TSO lockout or slow performance.

  With the default ADCD WLM parameters (at the time of writing), TSO is less important than started tasks. z/OSMF, for example, takes some time to start and consumes a large amount of system resources (CPU and I/O) during this time, making TSO usage rather poor during the z/OSMF start period. Customers with WLM configuration skills might want to experiment with this aspect.

- Temporary storage volumes.

  At the time of writing, the default z/OS ADCD system typically specified only one or two volumes (D4SYS1 and perhaps D4CFG1) for unassigned temporary data sets. In many cases, more volumes can help performance or avoid allocation issues. The easiest way to solve this issue is to update PARMLIB member VATLST00, for example:

  ```
  VATDEF IPLUSE(PRIVATE),SYSUSE(PRIVATE)
  D4SYS1,0,0,3390    ,Y
  WORK01,0,0,3390    ,N
  WORK02,0,0,3390    ,N
  ```

  This example assumes that your WORK01 and WORK02 volumes are suitable for temporary data sets.

12.2 IBM zEnterprise Data Compression emulation and the DFLTCC instruction

Several previous zPDT releases offered an option for emulating the IBM zEnterprise Data Compression (zEDC) function. The zEDC Express adapter on older IBM zSystems servers provided fast data compression or expansion. IBM z15 and IBM z16 machines replaced the adapter with the DFLTCC instruction, which provides even faster data compression or expansion. Since zPDT 1.11.1, the DFLTCC instruction is enabled by default.

Due to some implementation delays, ISV zPDT GA10 did not provide the DFLTCC instruction but continued to offer the zEDC function, even though zPDT GA10 is for the IBM z15 system architecture. ISV zPDT GA11 no longer offers the zEDC function. If a device map (devmap) that is used with GA11 specifies the zEDC parameters, a warning message is shown and the zEDC parameters are ignored.

ISV zPDT GA11 provides the DFLTCC instruction (in synchronous mode only), but ISV zPDT GA11 disables the facility indicator that software can check before using of the DFLTCC instruction. This change mitigates an obscure problem with the DFLTCC instruction that was present when ISV zPDT GA11 was first released.[11]

This distinction between the DFLTCC instruction and the facility might not be clear to all zPDT users. As a practical matter, most zPDT users do not directly code DFLTCC instructions or instructions to directly use a zEDC adapter. Instead, they allow internal z/OS functions (such as I/O access methods) to handle data compression and expansion. These z/OS functions can sense whether the DFLTCC facility is active; if a zEDC adapter is available; or if internal program code should be used to handle compression and expansion. Viewed a different way, the lack of a zEDC adapter or a disabled facility indication for DFLTCC might have a minor performance impact but would not greatly affect normal operations.

If you want to use directly the DFLTCC instruction, no special actions are needed. The instruction exists (in synchronous mode) in ISV zPDT GA11. However, users should be aware of the remote possibility of an obscure error in the instruction.

If the ISV zPDT GA11 user wants to enable normal z/OS usage of the DFLTCC instruction, use the zPDT command **`dflt`** to enable the facility indicator that is associated with the instruction. As a best practice, you should not use this approach, except in an experimental situation, until the problem with the DFLTCC instruction is resolved.

12.3 Maintenance for ADCD z/OS systems

For ISV zPDT users, a method is in place to allow bulk downloading of PTFs in a format that is equivalent to the Program Update Tapes (PUTs) that were formerly used for z/OS maintenance. (The PUTs are no longer in tape format, but the terminology remains.) We informally use the term “PUT” here, but only to refer to the material that is described in this section. These downloads are not provided by ISV zPDT, but instead they are provided by the ADCD team. In principle, anyone who has authority to download the z/OS ADCD volumes from the IBM Dallas site can also download the material that is described here.

[11] We expect that the problem will be corrected by later updates to zPDT GA11.

This optional material can be used for software maintenance for the products in the z/OS ADCD package. This material is for experienced z/OS systems programmers and not intended for casual ADCD users. Most ADCD users decide to obtain z/OS product maintenance through periodic ADCD releases. This facility might not be appropriate for all ISV zPDT users for two key reasons:

- The PUTs are large, typically up to 8 GB for each PUT. The PUTs are not cumulative, meaning that you might need to download and maintain multiple PUT releases.
- Substantial SMP/E skills are needed to use the materials. There are multiple ways that the material can be handled by the user and by SMP/E.

Each PUT is divided into several approximately 1 GB files on the download site. These files normally are concatenated for use by z/OS.[12] PUTs have names such as PUT1909, which stands for PUT number 9 for 2019.

A username and password are required. This username is unique to the Dallas download site and must be authorized for ADCD z/OS downloads. The Dallas group intends to maintain eight PUT levels on this site, but circumstances might change this number. This site provides a selection of various PUT levels, such as PUT1811. Selecting a PUT level lists the download files for that level, as in this example:

```
‘HTTPD1.fz206.f140.PUT1812.TRS10.BIN’
```

These PUTs can be downloaded through your browser download function. As a best practice, rename the files when downloading them.[13] One recent PUT file contains 999 PTFs in 78 million records, which is typical of recent PUTs. The large size of these files is usually due to large PTFs for UNIX System Services programs, especially for Java functions.

At the time of writing, the first file in a PUT level is small and contains only **ALIAS** statements for SMP/E. The remaining files tend to be large and are separated at PTF boundaries. After the multiple files for a PUT are downloaded to Linux, they should be transferred (with FTP) to z/OS files in a specific format. The receiving z/OS file (for the FTP transfer) *must* be RECFM=FB and LRECL=1024 (the block size does not matter). The FTP transfer must be binary. After transferring each file to z/OS, each must be processed by AMATERSE.

In our example, we found the easiest way to download the PUT files is to allocate a single receiving data set (RECFM=FB, LRECL=1024) for FTP. After each FTP completed, we ran an AMATERSE job to decompress the data set and assign it a logical data set name. The first data set in the PUT (in the TRS1 file) contains SMP/E **ALIAS** statements, and this data set is RECFM=VB, LRECL=255. The remainder of the data sets (containing the PTFs) is RECFM=FB, LRECL=80. The job that we used is as follows:

```
//UNPK  JOB  1,OGDEN,MSGCLASS=X
//  EXEC PGM=AMATERSE,PARM=UNPACK
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DISP=SHR,DSN=OGDEN.PUTUP
//SYSUT2   DD DISP=(NEW,CATLG),UNIT=3390,VOL=SER=WORK09,
//            DCB=(LRECL=80,RECFM=FB,BLKSIZE=8000),     for PTF files
//            SPACE=(CYL,(1500,50),RLSE),DSN=PUT1812.TRS2
//*           DCB=(LRECL=255,RECFM=VB,BLKSIZE=8000)     for TRS1 file only
```

[12] Another option is to combine the files into a single DSNTYPE=LARGE data set.

[13] Single quotation marks can be part of the file names on the download site. These single quotation marks must be "escaped" when used as Linux file names, and it is a best practice to avoid them.

We use FTP to transfer a file from Linux to `OGDEN.PUTUP` (which was allocated with 1500 cylinders, `RECFM=FB, LRECL=1024`). Then, we alter the AMATERSE job to have the correct output data set name, and run the job. We repeat this cycle for all the files in the PUT. The final result is a series of data sets, `PUT1812.TRS1`, `PUT1812.TRS2`, and so forth. For a different PUT level, we create a new high-level qualifier, such as PUT1901.

12.4 Java performance

z/OS Java performance can be an issue under ISV zPDT. One cause is the heavy system overhead (CPU and I/O) that is involved when starting some z/OS Java applications, especially applications that load many classes. The following notes might help improve performance a bit in some cases.

Important: Java details change frequently. The following notes might not be useful when you read this book.

- General base system

  Even simple Java applications tend to have heavy zFS I/O during start while the Java virtual machine (VM) loads the class libraries from the disk archives. If this I/O is contained mostly in the zFS cache or the Linux disk cache, general performance is improved. More PC memory can help.
- The raw performance of the base PC is an obvious concern, as is anything that can detract from performance, such as virtualization at the Linux level or at the z/VM level.
- Basic Java tuning

  At the time of writing, Java implementations use SIMD ("vector") instructions to accelerate string, crypto, and arithmetic operations on "real" IBM zSystems machines. The performance of these instructions varies with ISV zPDT, and in some cases Java performance might be better if the SIMD instructions are not used. To not use these instructions with some Java versions, include the statement **`-Xjit:disableAutoSIMD`** in the `jvm.options` file that is used for your Java application.
- IBM Rational Team Concert® usage

  Local area network (LAN) performance between an Rational Team Concert client system (that is, a separate PC) and the Rational Team Concert server (z/OS) depends on LAN speed.
- RESOLVER timeouts

  With some earlier z/OS Java levels, an unexpected Java performance problem can occur if you enable the z/OS RESOLVER, specify one or more external name server IP addresses, and the RESOLVER fails to connect to the name servers. In this case, any Java application that started (including javac) hangs until the RESOLVER experiences a timeout trying to connect to the name servers. This situation is true even for Java applications that have no LAN or TCP/IP usage. The length of the application hang depends on the timeout parameter in the RESOLVER, but is typically something like 60 seconds.

  This issue can be resolved by not running the RESOLVER or not specifying external IP addresses of name servers for it, or by verifying that RESOLVER connects to a working name server. With some Java releases, the JVM option **`-XX:-ReadIPInfoForRAS`** instructs the JVM to not query IP address information during startup, which bypasses this RESOLVER issue.

12.5 Larger 3270 display

The x3270 terminal emulator can be started with an optional parameter, for example:

```
$ x3270 -port 3270 -oversize 133x60 localhost &
```

This command produces a 3270 window with 133 columns and 60 lines. (Other sizes can be specified.) Basic TSO does not use the “extra” window width. ISPF can use it if `max` is specified for the window format in the ISPF option 0 panel.[14] (ISPF does not use the extra width unless a data set that is displayed or edited has records that can use the extra width.)

Another method is to define a model 5 3270 terminal is by running this command:

```
$ x3270 -model 3270-5 localhost:3270 &  (ampersand not needed if in devmap)
```

However, the “standard” 3279-5 has only 27 lines.

12.6 z/OS disk STORAGE volumes

In some circumstances, z/OS functions use STORAGE volumes[15] for disk allocations. The distributed z/OS ADCD systems generally have only a single STORAGE volume, which usually is named xxSYS1. As a best practice, for anything beyond trivial z/OS usage, define and mount *at least* one more 3390 volume as a STORAGE volume. A STORAGE volume is created by statements in the VATLSTxx member in PARMLIB. For example:

```
VATDEF IPLUSE(PRIVATE),SYSUSE(PRIVATE)     <===default values; do not change
B5SYS1,0,0,3390    ,Y
GEORGE,0,0,3390    ,N           <==we added this
WORK* ,0,0,3390    ,N           <==we added this
```

The format is fixed, and the spaces that are shown are important. Briefly, the parameters for volumes are:

- The volume serial number, in six columns. An asterisk matches any volser with the specified characters.
- The first ‘0’ is the mount attribute, which is normally set to ‘0’ unless there are unusual circumstances.
- The second ‘0’ is the use attribute. It is important. The value ‘0’ defines a STORAGE volume. Value ‘1’ defines a PUBLIC volume, and value ‘2’ defines a PRIVATE volume. The **VATDEF** statement sets the default use attribute to PRIVATE.
- The device type, in eight columns, left-aligned, and padded with blanks.
- An indicator whether z/OS should issue a MOUNT request for the volume. The value ‘Y’ indicates that z/OS requests that this volume will be mounted at IPL time if it is not mounted. Generally, ‘N’ is appropriate for local STORAGE volumes.

[14] You might need to scroll down in the ISPF option 0 panel to see the appropriate options.

[15] Generally, a STORAGE volume is used when creating a permanent data set and no volume is specified, although the exact details that are involved can be a little more complex.

The placement of SVC dump data sets can create a problem because they typically go to STORAGE volumes that, by ADCD default, is the xxSYS1 volume.[16] An MVS command can be used to direct SVC dumps to new volumes. The command is like `DUMPDS ADD,VOL=(volser,volser,volser)`. Much more sophisticated controls are available through SMS functions, but they require administrative work to create the environment.

12.7 Stand-alone z/OS dump

The z/OS stand-alone dump (SAD) is a program that starts (undergoes an IPL) from tape or disk. It does not run under z/OS. However, it assumes that z/OS is present in IBM zSystems memory at the time SAD is started. The SAD program is sensitive to the release of z/OS that is being used, and it must match the z/OS release. A new version of the SAD program should be generated whenever a new release of z/OS is installed.

This section describes a simplified usage of SAD based on ADCD z/OS 2.4, but the comments should apply to any recent z/OS release. The dump program is placed on a disk volume that can undergo an IPL, and the dump output data *must* be directed to a different disk volume. The SAD program (and output) can be used with tape volumes.

Disk volumes

Two disk volumes are referenced here:

- One volume is for the dump program itself, which is relatively small (about 100 tracks). This volume also contains IPL text to start the SAD program. This volume *must* be mounted as PRIVATE to run the dump generation job, but it can be in any mount status when performing an IPL of the dump program. An additional rule is that the SAD IPL volume cannot contain a paging data set for the system being dumped.

  Many z/OS ADCD users place the SAD program on a system volume, such as C4CFG1. For our example, we created two 3390 volumes: DUMPPP (200 cylinders) at address AD0 and DUMPDD (3390-9) at address AD1. The SAD program volume (AD0 in our example) is not sensitive to its address.
- The second volume is for the dump itself.[17] A SAD can be large, with hundreds of cylinders up to many thousands of cylinders. In our example, we assume that a suitable volume is mounted at address AD1. Before using the SAD program, you *must* create the dump output data set on this volume by using an IPCS or REXX utility function. The SAD data volume is sensitive to the address that you select (AD1 in our example) and must stay at the address that you specify.

The dump program (on which you perform an IPL) *cannot* be on the same volume that receives the dump data.

[16] You can look for data sets with names such as SYS1.ADCD.DMPnnnnn (or something similar, depending on the ADCD z/OS release) and delete them (assuming that you are not working on a problem that involves these dumps). The exact data set name pattern for SAN Volume Controller dumps is set in the appropriate COMMNDxx member in PARMLIB.

[17] The dump output can be directed to tape instead of a disk volume. Our discussion here concentrates on disk output.

12.7.1 Generating a stand-alone dump program

The following job generates the SAD program and writes it on volume DUMPPP. You later perform an IPL from this volume when you want to take a SAD.

```
//SADBILL JOB 1,OGDEN,MSGCLASS=X,REGION=40M
//         EXEC PGM=AMDSAOSG
//SYSLIB   DD SYS1.MACLIB,DISP=SHR
//         DD SYS1.MODGEN,DISP=SHR
//DSFSYSIN DD DSN=&DSFSYSIN,DISP=(,PASS),
//        SPACE=(80,(9,1)),UNIT=SYSDA
//TRKOTEXT DD DSN=&TRKOTEXT,DISP=(,PASS),
//        SPACE=(4096,(4,1)),UNIT=SYSDA
//GENPRINT DD SYSOUT=*
//GENPARMS DD *
  AMDSADMP IPL=D3390,VOLSER=DUMPPP,CONSOLE=(700,3279),OUTPUT=0AD1
         END
/*
//PUTIPL   EXEC  PGM=ICKDSF
//IPLDEV   DD   DISP=OLD,UNIT=3390,
//        VOL=(PRIVATE,RETAIN,SER=DUMPPP)
//TRKOTEXT DD  DSN=&TRKOTEXT,DISP=(OLD,DELETE)
//SYSIN    DD  DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD SYSOUT=*
```

Carefully check the Job Entry Subsystem (JES) 2 output when the job ends to ensure that it completed correctly. This job is unusual. If there are error messages (perhaps about PUTIPL SYSIN problems or a message from ICKDSF), you *must* delete the dump program data set (on volume DUMPPP in this example) before trying the job again. The data set probably has the name `SYS1.PAGEDUMP.VDUMPPP`. The format of the GENPARMS input conforms to basic assembly language rules, with the continuation indicator in column 72 and the continued text starting in column 16.

If you wrote IPL text on the dump program volume (DUMPPP in this example), there is an operator message that must be replied to before the IPL text is replaced. This volume (DUMPPP in the example) must be mounted PRIVATE when this job is run.

The error messages (or nonzero return codes) that appear when you attempt to run this job are not always helpful. There appears to be three common problems:

- The target volume for the dump program (the “IPL volume”) must be mounted PRIVATE while installing the dump program.
- The dump program and the target for the dump output cannot go on the same volume. You always have at least two disk volumes that are involved. In the example job, the IPL text and dump program are placed on volume DUMPPP. The dump output is placed on the volume at address AD1. This volume must have preallocated space for the dump output.
- The dump program (on the IPL volume that you designate) must be deleted before you can try the job again.

12.7.2 Stand-alone dump output data set

Output ("the dump") from the Stand-Alone Dump program requires a preallocated data set. The data set can be created with IPCS or with a REXX program that is named AMDSADDD. Briefly, the REXX program can be used by going to ISPF option 6, entering "`AMDSADDD`", and replying to the prompts as follows:

```
What function do you want .....
define
Enter the volume serial ....
DUMPDD                        (Assume that DUMPDD is at address AD1 in our
                               example.)
Enter device type .....
3390
number number of cylinders.....
6000                          (this might be a reasonable size...)
Do you want the dataset cataloged...
n
Specify the DSNTYPE. Reply BASIC, or LARGE or EVTREQ...
large
Specify additional attributes
no

Data set SYS1.SADMP is allocated on volume DUMPDD
```

As described here, the dump data set can be used for only one dump. To reuse it, you must reset or clear it. To do so, use the same AMDSADDD program and select a different option at the first prompt.

12.7.3 Operating a stand-alone z/OS dump

We assume that you are running z/OS and need a SAD for some reason. To do so, run the following commands:

```
$ stop all               (Stop the CPs.)
$ ipl AD0                (Assume that volume DUMPPP is mounted at this address.)
```

Wait about 10 seconds, and then press Enter on the 3270 console at address 700. This action produces console messages like the following ones:

```
AMD083I AMDSADMP: STAND-ALONE DUMP INITIALIZED. IPLDEV: 0AD0 LOADP:
AMD001A SPECIFY OUTPUT DEVICE ADDRESS (1):    (press Enter)
AMD101I OUTPUT DEVICE: 0AD1
        SENSE ID DATA: FF 3490 10 3490 40  BLOCKSIZE: 29,120
AMD011A TITLE=your dump stuff
AMD005I DUMPING OF READ STORAGE NOW IN PROGRESS
AMD005I DUMPING OF PAGE FRAME TABLE COMPLETED
    etc
AMD029D REPLY W TO WAIT AFTER NEXT FULL SCREEN, ELSE REPLY N; REPLY=n
    etc
```

12.8 Moving 3390 volumes

The following section describes a generic method of moving 3390 volumes between z/OS systems (including z/OS on ISV zPDT). Another method, which uses a unique client/server application that is provided with ISV zPDT, is described in Chapter 15, “Direct access storage device volume migration” on page 313.

ISV zPDT emulated direct access storage device volumes can be transferred to other systems in several ways. Sending a volume to another ISV zPDT system is especially easy. The Linux file that holds the emulated 3390 volume can be copied.[18] Optionally, the copy can be compressed (with **`gzip`**, for example) for transmission. The transmission can be made by FTP, a USB flash drive, burning a CD or DVD, or by various other means. The key concept is that a large Linux binary file is transferred.

Moving an emulated 3390 volume to (or from) a non-ISV zPDT system is more complex because it must be handled in an IBM zSystems format instead of a Linux format. The traditional method is to dump the volume to tape (by using the ADRDSSU program) and then restore the tape on the target system. This method also can be used with emulated tapes (in awstape format) if both the sending and receiving systems can use this format.

The following example assumes that both systems cannot use awstape format (otherwise, we would use the easier method of creating a dump tape in awstape format). We assume that there is a network connection between the source z/OS and the target z/OS system. We also assume that the target system is an ISV zPDT system, although that is not a requirement for the technique that described.

Figure 12-1 shows an overview of our example.

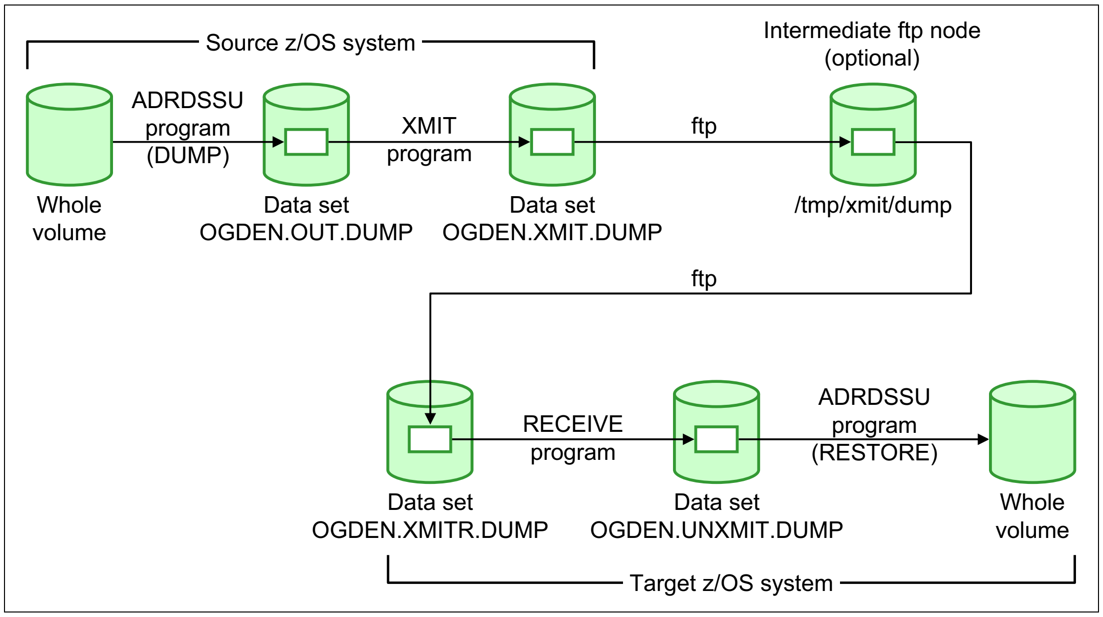


Figure 12-1 Overview of our example

[18] Do this task when z/OS is not running if the volume is included in the “running” devmap.

Preparation

We need z/OS disk space to hold a 3390 volume dump and a reformatted volume dump. These dumps can be large data sets. You might have sufficient space on existing z/OS volumes, but we decided to create three volumes for holding large temporary data sets. We did this task by using normal ISV zPDT techniques.

1. We created three emulated 3390 volumes by using the **alcckd** command while ISV zPDT was not operational. The placement (/z directory), model (3390-3), and names of the files (TEMP*nn*) are all arbitrary.

   ```
   $ alcckd /z/TEMP01 -d3390-3
   $ alcckd /z/TEMP02 -d3390-3
   $ alcckd /z/TEMP03 -d3390-3
   ```

2. We added these volumes to our devmap. The addresses that are specified (AA0, AA1, and AA2) are unused addresses that are known as 3390 devices for our z/OS. (That is, z/OS specifies these addresses as 3390 devices in the input/output definition file (IODF) that it uses during IPL. The AA*x* addresses are suitable for the default IODF in the z/OS AD systems.)

   ```
   [manager]
   name awsckd 0001
   ....
   ....
   device AA0 3390 3990 /z/TEMP01
   device AA1 3390 3990 /z/TEMP02
   device AA2 3390 3990 /z/TEMP03
   ```

3. We started ISV zPDT and performed an IPL on z/OS. During z/OS start, the new devices are recognized as uninitialized volumes and varied offline. When z/OS was ready, we ran a job to initialize the volumes:

   ```
   //BILL123 JOB 1,OGDEN,MSGCLASS=X
   //  EXEC PGM=ICKDSF,REGION=40M
   //SYSPRINT DD SYSOUT=*
   //SYSIN DD *
    INIT UNIT(AA0) NOVALIDATE NVFY VOLID(TEMP01) PURGE -
      VTOC(0,1,05)
   INIT UNIT(AA1) NOVALIDATE NVFY VOLID(TEMP02) PURGE -
      VTOC(0,1,05)
   INIT UNIT(AA2) NOVALIDATE NVFY VOLID(TEMP03) PURGE -
      VTOC(0,1,05)
   /*
   ```

The z/OS operator must reply U to a ICK003D message for each volume. The volsers (TEMP01, and so forth) are the same as the Linux file names, which is not required, but it is a best practice. After the volumes are initialized, they can be varied online to z/OS by using the MVS console:

```
vary aa0-aa2,online
```

12.8.1 Creating a source dump

A normal ADRDSSU job is used to dump the source volume. Our example uses WAS001 as the volser of the source volume:

```
//BILL456 JOB 1,OGDEN,MSGCLASS=X
//  PGM=ADRDSSU,REGION=40M
//SYSPRINT DD SYSOUT=*
//IN DD UNIT=3390,VOL=SER=WAS001,DISP=SHR
//OUT DD UNIT=3390,VOL=SER=TEMP01,DISP=(NEW,CATLG),
//       DSN=OGDEN.OUT.DUMP,SPACE=(CYL,(200,200))
//SYSIN DD *
  DUMP INDD(IN) OUTDD(OUT) ADMINISTRATOR COMPRESS OPTIMIZE(4)
/*
```

The space that is specified in the output **DD** statement might need to be adjusted depending on the contents of the source volume. We created another data set with specific dataset control block (DCB) attributes.[19] (This step can be done by using ISPF 3.2 functions, but we used a batch job to provide better documentation.)

```
//BILL567 JOB 1,OGDEN,MSGCLASS=X
//  PGM=IEFBR14
//MAKEIT DD UNIT=3390,VOL=SER=TEMP02,DISP=(NEW,CATLG),
//  DSN=OGDEN.XMIT.DUMP,SPACE=(CYL,(200,200)),
//  DCB=(LRECL=80,RECFM=FB,BLKSIZE=3120)
```

We used the TSO **xmit** command to reformat the dump:

```
xmit x.y ds('ogden.out.dump') outdsn('ogden.xmit.dump')
```

This command can take considerable time if a full volume is being processed. The result of these steps is a volume dump in a format that is known to z/OS, but also in a format (fixed block) that can be handled by FTP. The **x.y** positional operand is needed in **xmit**, but it is meaningless in this example. The **terse** program can be used instead of **xmit**.

It is important to understand the reason for the **xmit** step. In general, FTP does not understand the block and record structure of a z/OS file (such as the ADRDSSU output file). This information is lost during FTP and the resulting file is not usable. The **xmit** program changes the ADRDSSU dump file into a fixed block, fixed record format. The general FTP process does not understand this task either. However, if the transferred (with FTP) file is stored in the receiving z/OS with the same fixed block and LRECL size, the file is usable.

Some FTP situations allow extra parameters such that the original block and record characteristics of a file are retained and the **xmit** step can be skipped. This task can be done in a z/OS to z/OS FTP transfer. The method that is presented in this section assumes the more general case in which the FTP transfer does not retain the original block/record information.

[19] These DCB attributes are used by XMIT.

12.8.2 Sending a dump to Linux

We sent the **xmit**-formatted dump to Linux by using an FTP connection from Linux to z/OS. We used the ISV zPDT tunnel facility for the connection in our example, but any TCP/IP connection to z/OS can be used. The IP address for z/OS is `10.1.1.2` in this example:

```
$ ftp 10.1.1.2
Name (10.1.1.2:ibmsys1): ibmuser
Password: xxxxxx
Remote system type is MVS
ftp> cd 'ogden'
ftp> lcd /tmp
ftp> bin
ftp> get 'xmit.dump'
ftp> bye
```

This example has an FTP connection that is initiated from the Linux side. This action can be done from the z/OS side if the Linux system has an FTP server running.

The dump is now in `/tmp/xmit.dump` as a normal (large) Linux file. It can be transmitted elsewhere by using any technique that is suitable for a large Linux file. The file can be compressed (by using **gzip**, for example.) Now, the dump (`OGDEN.OUT.DUMP`) and the reformatted dump (`OGDEN.XMIT.DUMP`) on the source z/OS system can be deleted if disk space is a concern.

You can skip this intermediate Linux step if there is a direct FTP connection between the source z/OS system and the target z/OS system.

12.8.3 Receiving the dump

There are fewer complications if two data sets are preallocated on the receiving z/OS system: One data set is the target of an FTP transfer from Linux (or some other source), and the other data set is for the output of the **TSO RECEIVE** function. This last data set is the input to a RESTORE job.

```
//BILL678 JOB 1,OGDEN,MSGCLASS=X
//  PGM=IEFBR14
//D1 DD UNIT=3390,VOL=SER=TEMP01,DISP=(NEW,CATLG),
//   SPACE=(CYL,(200,200)),DSN=OGDEN.XMITR.DUMP,
//   DCB=(LRECL=80,RECFM=FB,BLKSIZE=3120)
//D2 DD UNIT=3390,VOL=SER=TEMP02,DISP=(NEW,CATLG),
//   SPACE=(CYL,(200,200)),DSN=OGDEN.UNXMIT.DUMP
```

The dump file can be sent from Linux to z/OS by using FTP. (If the file was compressed in Linux, it must be decompressed before sending it to z/OS.) Our example uses IP address `10.1.1.2` for z/OS because for demonstration purposes, we used the same ISV zPDT z/OS system that we used to create the dump volume. In practice, this system is likely to be a different z/OS system that might not be in an ISV zPDT environment.

```
$ ftp 10.1.1.2
Name (10.1.1.2:ibmsys1): ibmuser
Password: xxxxxx
Remote system type is MVS
ftp> cd 'ogden'
ftp> lcd /tmp
ftp> bin
ftp> put xmit.dump xmitr.dump
ftp> bye
```

We used TSO to reformat the dump into the original format that was created by ADRDSSU:

```
receive indsn('ogden.xmitr.dump')
```

Reply to the prompt with `DSN('ogden.unxmit.dump')`.

Finally, the volume can be restored in z/OS:

```
//BILL890 JOB 1,OGDEN,MSGCLASS=X
//  PGM=ADRDSSU,REGION=40M
//SYSPRINT DD SYSOUT=*
//IN DD UNIT=3390,DSN=OGDEN.UNXMIT.DUMP,DISP=SHR
//OUT DD UNIT=3390,VOL=SER=TEMP03,DISP=OLD
//SYSIN DD *
  RESTORE INDDNAME(IN) OUTDDNAME(OUT) PURGE ADMINISTRATOR COPYVOLID
/*
```

You might not want the **`COPYVOLID`** parameter in this job, depending on your circumstances. You cannot have two disk volumes with the same volser online to z/OS at the same time. If you do not specify **`COPYVOLID`**, the existing volser (TEMP03 in the example) is retained. If you specify **`COPYVOLID`** and a volume with this volser is already online, the restored volume is taken offline after the restore operation is complete. (In our example, the restored volser would be `WAS001`.)

Comments

Many variations are possible in this general process. For example, some of the z/OS preallocation of data sets can be skipped if your FTP supports **`site`** and **`locsite`** subcommands.

12.9 IODF changes with ISV zPDT

When working with a larger IBM zSystems system, IODF and IOCDS creation are almost always done at the same time while working with HCD. This process typically starts as follows:

```
HCD (usually started from an ISPF panel)
      1. Define, modify, or view configuration data
         3. Processors
         4. Control Units
         5. I/O Devices
```

The HCD functions verify that an allowable configuration is specified, that is, the processor (type and model), CHPIDs, and I/O devices must all be mutually allowable. This check is useful for a larger IBM zSystems system, but it creates problems with ISV zPDT.

An ISV zPDT system does not use an IOCDS and does not understand many hardware CHPID details. Furthermore, typical ISV zPDT configurations are not compatible with the normal HCD verification and processing that you use with a larger IBM zSystems system. At the time of writing, I/O device configuration for an ADCD z/OS system on ISV zPDT consists of a *software-only* IODF generation, which is followed by the creation of a matching devmap. Many z/OS users are not familiar with a software-only IODF, so we provide an overview of the topic here. It is considerably simpler than a “normal” IODF.

Assume that we want to add 16 OSA devices starting at address 410. Using the IODF99 that is provided with the current z/OS ADCD system[20], we can proceed as follows:

```
HCD                        (Started through ISPF menu item M.4)
     1. Define, modify, or view configuration data
                  I/O DEFINITION FILE 'SYS1.IODF99'
       1. Operating System Configuration
         / OS390           (select configuration named 'OS390')
           7. Work with attached devices
                           (This should produce a list of current devices)
          F11 - Add
```

Now, you should have a panel in which you can enter a new work IODF name. We entered the name `SYS1.IODF77.WORK` and volser `C2SYS1` in this panel. The data set name should follow this pattern (although the `77` portion of the name is arbitrary) and the volser should be the volume that is specified in the IPL parameter (`C2SYS1` in the z/OS 2.2 AD system).

A panel opens where you can add devices to the new work IODF. We entered the following information:

```
Specify or revise the following values.

Device number. . . . . . . 410    + (0000 - FFFF)
Number of devices  . . . .  16
Device type. . . . . . . . OSA    +
Serial number. . . . . . .
Description. . . . . . . .
Volume serial number . . . ______  (for direct access storage device)
Connected to CUs . . ____ ____ ____ ____ ____ ____ ____ ____
```

Press Enter and again select (with a **/** character) the OS390 configuration. Select option 1 (to connect or change the new I/O devices). A panel opens where you can alter the default device parameters. Use the default options unless you have a reason for changing them. Press Enter. A panel opens where you can associate the esoteric names with the new devices. You might use this item for direct access storage device or tape devices, but probably not for any other types of devices. Press Enter and then select (with a **/** character) the OS390 configuration again.

Press F3 to return to the device list, and you should now see 410,16 in the list. When your new I/O devices are added to the list, use F3 three times to return to the initial HCD menu. Then, you process your new IODF file:

```
2. Activate or process configuration data
               I/O DEFINITION FILE 'SYS1.IODF77.WORK'
  1. Build production I/O definition file
               (This can produce some warning messages, usually about esoteric
               tokens. Ignore these messages. F3 to exit the warning panel.)

Command ==>

Date & Time . . . . . . : 2008-07-14  13:14:51
User . . . . . . . . . .: IBMUSER
I/O Definition file. . .: SYS1.IODF77.WORK
Change reference number.: 00018
****** ************************TOP OF DATA *************************
.......
```

[20] This example is based on the z/OS 2.2 ADCD system, but later releases should be similar.

```
****** ************************BOTTOM OF DATA **********************
```

Press F3 to exit this panel. In the next panel, you name the new production IODF file:

```
Specify the following values, and chose how to continue.
Work IODF name . . . . . . : 'SYS1.IODF77.WORK'
Production IODF name . .: 'SYS1.IODF77'
Continue using as current IODF:
1   1. The work IODF in use at present
    2. The new production IODF specified above  (not valid for ISV zPDT)
```

In the next panel, specify or revise these values, and then press Enter. The message `PRODUCTION IODF SYS1.IODF77 CREATED` should appear. Press F3 several times to exit from the HCD.

To use the new IODF, you must alter one or more of the LOADxx members in `SYS1.IPLPARM` to refer to the new IODF. For example, edit member `LOAD00` in `SYS1.IPLPARM`:[21]

```
IODF     99 SYS1              <--change this line
SYSCAT   Z9SYS1113CCATALOG.Z19.MASTER
SYSPARM  00
IEASYM   00
NUCLST   00
PARMLIB  USER.PARMLIB                             Z9SYS1
PARMLIB  ADCD.Z112.PARMLIB                        Z9RES1
PARMLIB  SYS1.PARMLIB                             Z9RES1
NUCLEUS  1
SYSPLEX  ADCDPL
```

Change the `99` in the first line to `77` (or whatever number that you used for your IODF). The format of this statement is odd, but it results in the name `SYS1.IODF77`. Be certain to place your changed characters in the same columns as the original characters. Do not change anything else in the LOADxx member unless you are certain about your actions.

The new IODF is now ready to use the next time that you perform an IPL for z/OS with the parameter:

```
$ ipl 0a80 0a8200              (The 00 corresponds to the LOAD00 member)
```

Assuming that you are satisfied with the results, you probably want to change all the LOADxx members that you use. You also must change your devmap to use the new devices that you added to your z/OS system.

[21] This example is from z/OS 1.9 and 1.12. Other details in your LOADxx member will differ from this example. The key point here is in the first line of the member.

12.10 Local printing

Hardcopy printed output is rare in today's working environments, but it is sometimes needed. There are many ways to accomplish this task. The following section describes one of these ways.

Background

Basic z/OS printing is closely related to the hardware that was available on IBM S/360 machines. The most common printer then was the IBM 1403. It printed lines with 120 characters (or 132 characters with an optional feature) and was normally set to print 6 lines per inch on fan-fold paper that was 11 inches long. This configuration produced a full page that held 66 lines. In practice, many programs counted output lines and skipped to a new page after 60 or 61 lines.

Much of the utility software with the system, such as JCL processors, assembler languages, compilers, and system report programs were designed to fit these pages. They printed lines of up to 120 characters (sometimes up to 132 characters) with about 60 lines per page. This default convention is still with us today.

Later hardware replaced the line printers (such as the 1403) with laser printers. They were devices such as the IBM 3800, 3820, 3825, and 3900. They accepted many paper sizes, but were most commonly used with "letter size" paper.[22] With proper programming, these printers produced sophisticated output by using many font and graphic components. However, the system utilities (compilers, for example) continued to produce listings in "1403 format." Software for these laser printers can accept this 1403-format data and list it. These listings typically are two-sided, landscape mode, and contain up to 66 lines of 132 characters on each page.

Real 1403 printers are historical items, but there are many uses for pseudo-1403 devices. z/OS and JES2 still support 1403 printers, and ISV zPDT can emulate 1403 printers.

Using a PC printer

In this example, our goal was to use a common PC laser printer and have utility output that was produced in the format that was described: landscape mode, 66 lines per page, and 132 characters per line. If the PC printer provided duplex printing (printing on both sides of the paper), it was used. The flow is illustrated in Figure 12-2.

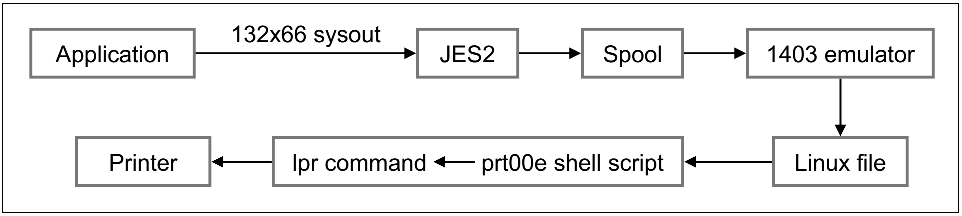


Figure 12-2 General flow for printing

Our tests used Lexmark OptraS1250 and Optra L printers (both with duplex printing features). We did not try the techniques that are described here with other printers, but we expect the same or similar techniques can be used. However, z/OS printed output typically contains separator pages, JCL listings and messages, and so forth, so the smallest job usually has multiple pages of printed output. This approach might not be suitable for use with a small inkjet printer. We assumed the usage of a heavy-duty laser printer for z/OS printing.

[22] The "letter size" (or A4) paper can be cut sheets or fanfold paper, depending on exactly which printer is being used.

12.10.1 Setup

In this section, we provide the setups for Linux, the ISV zPDT devmap, a shell script, and JES2.

Linux setup

We configured our (old) Lexmark Optra S1250 for Linux. This printer had a parallel input, and our computer had no parallel ports. We purchased a USB-to-parallel cable to provide the necessary connectivity. We used YAST (running under openSUSE) to configure the printer.[23] A print queue that is named optras1250 was created automatically by YAST. We verified that the printer worked by using the following commands:

```
$ lpr /home/ibmsys1/prof12              (to print one of our devmaps)
```

Devmap setup

We added the appropriate 1403 definition to the ISV zPDT devmap:

```
[manager]
name awsprt 4321 --windows
device 00E 1403 2821 /tmp/1403a
```

The **--windows** option was needed to place CR/LF characters in the output because without this option, New Line (NL) characters would be used and a PC printer might not work with NL characters. The output file name (`/tmp/1403a` in the example) was arbitrary. In our case, we did not expect much printed output, and `/tmp` seemed a reasonable place to put the output. The output file can also be assigned or changed with the **awsmount** command.

Shell script

We created a Linux shell script that is named prt00E and placed it in our home directory (`/home/ibmsys1`, in our case). The shell script contained the following lines:

```
CTL=”\033\105\033\050\163\060\160\061\066\056\066\067\150\070\056\166
     \060\163\060\142\124\033\046\154\061\157\061\163\065\056\064\143
     \055\061\060\060\060\132”
CTL2=”\014\033\105”
(/bin/echo -ne $CTL; cat /tmp/1403a; /bin/echo -ne $CTL2) | lpr -P optras1250
```

The CTL and CTL2 definition constants are printer control characters that are written in octal.[24] The particular control characters that are shown here are for the Lexmark printers. Your printer might require different controls. If you are printing to the default Linux printer, you do not need the **-P** parameter (and queue name) of the **lpr** command.

The logic in the shell script is simple. It sends data (through a pipe and **lpr**) to the queue that we defined earlier. It sends the `CTL` string (by using **echo**); sends the print data from the output file that we named in the devmap or **awsmount** command (by using **cat**); and then sends the `CTL2` string (by using **echo**) to flush the printer buffer and reset the printer.

The `CTL` control string (for our Lexmark printers) resets the printer; switches to a fixed font; sets a small type size pitch; changes to 8 lines/inch; uses a Courier font; uses landscape, duplex printing; uses 5.4/48 line height; and uses a small line offset to better center the data. The only unique requirement is that the format must use *exactly* 66 lines per page to synchronize with the pages that are produced by the 1403 emulator.

[23] Red Hat Linux has a different configuration process, but the results are the same.

[24] `CTL` is shown as three lines here, but it is created as one long line containing 38 octal constants.

In the following strings, `\033` is the ESC character that is used to begin printer command strings, which is shown as a bold-face **E** in the character equivalents of the string. The octal constants are the equivalent of the characters that are shown. The `CTL` string can be written with characters (except for the ESC byte).

The `CTL` commands are as follows:

```
OCTAL constant.......            Characters  Comment
\033\105                            EE       Reset printer
\033\050\163\060\160                E(s0p    Use a fixed font,
\061\066\056\066\067\150            With 16.67-inch character pitch
\070\056\166                        8.v      with 8 lines/inch characteristics
\060\163\060\142\124                0s0bT    Using upright, medium Courier
\033\046\154\061\157                E&l1o    Use landscape format
\061\163                            1s       with duplex printing, long edge
\065\056\064\143                    5.4c     5.4/48 inch line height
\055\061\060\060\060\132            -1000Z   line offset
```

The `CTL2` commands are as follows:

```
\014\033\105                                 Force last page, reset printer
```

The `CTL` string was produced after some experimentation. It works for our printers, but it might contain unnecessary elements. We hardcoded the file name (`/tmp/1403a`) in the shell script, but more skilled users might want to make this file name a CLI variable.

JES2 setup

Normal z/OS printing flows through JES2, and the printers must be known to JES2. Recent ADCD systems do not have a printer that is defined for JES2. You must define a 1403 printer at address 00E for JES2. (We use device number 00E because it is the traditional address for a 1403 printer and already is defined in the ADCD IODF.) Edit the ADCD PARMLIB member JES2PARM to contain the following line:

```
PRT(1)  WS=(W,R,Q,PMD,LIM/F,T,C,P),UNIT=00E,CLASS=C
```

If you scroll through the existing JES2PARM (in the ADCD system), you might find commented lines like this one. You can insert a new line, as shown, or convert the commented lines into active lines.[25] The print class (`CLASS=C`) is arbitrary. We selected class C because nothing defaults to this class.

We added the printer definition to JES2 and did another IPL of z/OS. We verified that the printer was online (**`d u,,,00E,1`**) and then issued a JES2 command to start it (**`$SPRT1`**). Use a **`$SPRT1`** command as the response to requests to mount forms, and so forth.

12.10.2 Operational technique

We ran jobs that sent output to `SYSOUT=C`. For example:

```
//OGDEN1 JOB 1,OGDEN,MSGCLASS=C
//  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DSN=SYS1.PARMLIB(IEASYS00),DISP=SHR
//SYSUT2 DD SYSOUT=*
```

[25] If you convert the commented lines into active lines, be especially careful with the /*....*/ comment indicators, that is, be certain that you remove matching pairs.

After submitting this job, you should see z/OS console messages about jobs that are sent to PRT1. If JES2 requests a setup function for the printer, reply **$SPRT1**. If the emulated printer is started (for JES2), the printed output is immediately sent to the "printer." As described here, this file is file /tmp/1403a in Linux. Additional output (from multiple jobs) is added to the file. The emulated printer cannot distinguish where one job ends and the next begins, so the JES2 separator pages are needed for this task.

You can stop the JES2 printer (**$PPRT1**) and print the accumulated output under Linux as needed. (You do not need to stop the JES2 printer if you are certain no additional output will be sent to it.)

Disconnect the output file (/tmp/1403a) from the emulated printer:

```
$ awsmount 00E -u
```

Run the shell script:

```
$ cd /home/ibmsys1                  (Directory containing the shell script)
$ ./prt00E                          (Run the shell script.)
```

The printer begins printing output. The output includes all the job separator pages that are produced by JES2. When it finishes, you can use **awsmount** to provide an empty output file for the emulated printer:

```
$ rm /tmp/1403a                     (Delete the old output file.)
$ touch /tmp/1403a                  (Start a new output file with the same
                                    name.)
$ awsmount 00E -m /tmp/1403a        (Connect the new output file.)
```

We deleted the output file. We re-created the same file (because the name is hardcoded in the shell script) and "mounted" it on the emulated printer so that it is ready for more output.

If you stopped the JES2 printer, you need to start it again (**$SPRT1**).

12.11 Lost MVS console

If you accidentally close the TN3270e session that contains the MVS operator console, try the following recovery procedure.

Try to re-establish the session, which might be easier if the MVS console has an LUname in the devmap, as in this example:

```
$ x3270 -port 3270 mstcon@localhost &
```

Depending on exactly what was happening when the console was lost, the TN3270e connection to the aws3174 device manager might still be active, so you cannot make a "new" connection to it. You can force the console session to disconnect by running the following command in a Linux window:

```
$ awsmount 700 -d              (Assuming your MVS console is address 700)
```

You might be able to connect the TN3270e session to address 700. You must connect the TN3270e session before proceeding with more recovery.

When MVS lost the console, it probably started issuing Hardware Management Console (HMC) messages in the Linux window that is used to start ISV zPDT. You can attempt to reactivate the MVS console on 700 (assuming that you have a TN3270e connection to 700) as follows:

```
$ oprmsg v 'cn(*),activate'      (Activate the HMC for commands.)
$ oprmsg v 700,offline
$ oprmsg v 700,offline,force     (if the simple vary offline fails)
$ oprmsg v 700,online
$ oprmsg v 700,console
$ oprmsg v 'cn(*),deactivate'     (optional)
```

This approach might not always work. Also, it might produce a console in 3270-2 (24 lines) mode, which is better than no console. The single quotation marks in the commands are needed to prevent the Linux shell from directly using the parentheses characters in the commands.

12.12 Customized Offering Driver

The Customized Offering Driver (COD) is a small, preconfigured z/OS system that is delivered on a set of DVDs. The COD is intended as a base for installing a z/OS ServerPac or Custom-Built Product Delivery Option (CBPDO). The COD, at the time of writing, has the following characteristics:

- Three 3390-9 volumes are used, each distributed as a set of files on its own DVD. A separate DVD is included that contains documentation.
- A wide range of addresses (device numbers) for 3390 and local 3270 consoles are included, with a smaller range of addresses for Systems Network Architecture (SNA) 3270-X, 3590 (tape), OSA, channel-to-channel (CTC), and SCTC devices.
- TCP/IP is usable with some configuration work.
- Instructions are included for working as a z/VM guest.

Why would an ISV zPDT user want to use the COD? The reasons might include the following ones:

- You want to work with a more “basic” z/OS system than the ADCD z/OS system. The ADCD z/OS system contains a considerable amount of customization to make it easier to use. Systems programmer training might include starting with a much more basic z/OS level, and the COD might be a more appropriate starting point.
- A simple practice platform before using the COD on a larger IBM zSystems system.
- Few ISV zPDT users use the COD (or can obtain it), but using it on a simple ISV zPDT system can provide important experience for users who need this rather unique package.

The COD is not part of ISV zPDT deliverables and cannot be obtained through ISV zPDT or ADCD channels.

The following example is based on the SUSE Leap42.2 system. Your Linux might have slight differences, for example, the DVD drive might be named something other than `/dev/sr0`. The installation example here assumes that you are working with Linux user ID `ibmsys1`. We use **`su`** as root, but you can use **`sudo`** instead if your Linux system is configured for it.

Before starting, create a Linux directory that is named `/mnt/ibmdvd` that can be accessed by user `ibmsys1`:

```
# su                                        (Switch to root.)
# mkdir /mnt/ibmdvd                         (Create a mount point for COD DVDs)
# exit                                      (Leave root.)
```

Create three 3390 volumes. At the time of writing, the volsers were named D9ECAT, D9ESY1, and D9ESY2, and they are used in our example. We created our emulated volumes in directory `/z`.

```
$ alcckd /z/D9ECAT -d3390-9
$ alcckd /z/D9ESY1 -d3390-9
$ alcckd /z/D9ESY2 -d3390-9
```

We created an ISV zPDT devmap that is named devcod as follows:

```
[system]
memory 8G
processors 1            #You can specify three processors if they are available.
3270port 3270
command 2 x3270 localhost:3270

[manager]
name aws3274 0002
device 00A1 3270 3274           #MVS console
device 00C0 3270 3274           #TSO

[manager]
name awsckd 0004
device 0170 3390 3990 /z/D9ESY1
device 0171 3390 3990 /z/D9ESY2
device 0172 3390 3990 /z/D9ECAT
```

Each of the three 3390 volumes is installed in the same way. The following text illustrates installing the D9ECAT volume:

```
(Insert the D9ECAT DVD and assume that Linux will automount it.)
# su                                        (Switch to root.)
# umount /dev/sr0                           (Unmount the DVD.)
# mount -t iso9660 -o ro,map=o /dev/sr0 /mnt/ibmdvd
# exit                                      (Leave root.)
$ ls /mnt/ibmdvd                            (Should see CAT)
$ ls /mnt/ibmdvd/CAT                        (Should see multiple files)
$ awsstart devcod                           (Start ISV zPDT.)
$ ipl_dvd /mnt/ibmdvd/CAT/DFSMSDSS.INS      (Uppercase is important.)
$ Enter ‘y’ to continue.....
$ y                                         (Reply y. Wait a few seconds.))
$ Memory loaded. 0x20A0 Bytes at address 0x0
$ Memory loaded. 0x3FFC0 Bytes at address 0x20000
   (After a few seconds press Enter on the 3270 panel.)
   (You should see a message to clear the panel. Use only the 3270 clear
    operation.)
$ ADRY005E DEFINE INPUT DEVICE, REPLY ‘DDDD,CCUU’ OR ‘CONSOLE’
$ ENTER INPUT/COMMAND:
$ CONSOLE                                   (your response)
$ ADRY006E DEFINE OUTPUT DEVICE, REPLY ‘DDDD,CCUU’ or ‘CONSOLE’
$ ENTER INPUT/COMMAND:
```

```
$ CONSOLE                                  (your response)
$ ENTER INPUT/COMMAND:
$ RESTORE FROMDEV(DVD) TOADDR(0172) PATH('/CAT') FULL NOVERIFY
$ ADRY003D 0172 REPLY Y TO ALTER VOLUME, ELSE N
$ ENTER INPUT/COMMAND:
$ y                                        (your response)
   (After a pause, a number of progress messages should appear.)
$ FILE CAT001 has been processed with 0125806839 Bytes
  etc
  (Clear the panel if requested.)
$ ADRY0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
$ awsstop
```

We stopped ISV zPDT between restoring volumes to unmount the current DVD:

```
$ su
# umount /dev/sr0
# eject
```

We processed the other DVDs with the only change being the **ipl_dvd** and **RESTORE** commands:

```
ipl_dvd /mnt/ibmdvd/SY1/DFSMSDSS.INS
RESTORE FROMDEV(DVD) TOADDR(0170) PATH('/SY1') FULL NOVERIFY
ipl_dvd /mnt/ibmdvd/SY2/DFSMSDSS.INS
RESTORE FROMDEV(DVD) TOADDR(0171) PATH('/SY2') FULL NOVERIFY
```

The correct **PATH** operand is critical. The **restore** command can be entered in lowercase if you prefer. Uppercase or lowercase is critical for Linux commands, but they usually are ignored for z/OS commands, including the stand-alone utilities.

After restoring the COD, we added a second command statement to the devmap to automatically start a second 3270 session:

```
command 2 x3270 -model 3279-2 localhost:3270
```

The IBM Virtual Telecommunications Access Method (VTAM) modtabs that are provided with the COD support only 24x80 sessions for VTAM and TSO. If we use default x3270 panels (43x80), it is easy to mistake a partly filled panel (which can be scrolled forward) for the end of a list. The MVS console automatically adapts to the 43x80 panel size.

After starting ISV zPDT, we proceeded as follows:

```
$ ipl 0170
```

After a few seconds, the MVS console interaction (with the 3270 at address 00A1) was as follows:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR zSystems/OS 02.01.00 HBB7790
r 00,clpa,sysp=00
....messages...
01 ICH502A SPECIFY NAME FOR PRIMARY RACF DATASET SEQUENCE 001 OR 'NONE'
r 01,SYS1.RACF
02 ICH502A SPECIFY NAME FOR BACKUP RACF DATA SET ....
r 02,NONE
....lots of messages....
03 $HASP426 SPECIFY OPTIONS - JES2 z/OS 2.1
r 03,COLD,NOREQ
   (you might have a message to allow bypassing the multi-member
    integrity lock.  If so, reply Y.)
04 $HASP441 REPLY 'Y' TO CONTINUE INITIALIZATION ......
r 04,Y
...lots of messages....
   (Wait until you see the OMVS INITIALIZATION COMPLETE message)
d u,,,0C0                              (Verify that the TSO terminal is online.)
s vtam                                 (Start VTAM and wait a few seconds.)
s tso                                  (Start TCAS.)
v net,act,id=d0c00df                   (appropriate for address 0c0)
```

The `d0C00df` in this command is the name of a VTAMLST member that is related to local 3270 address 0C0 (this information is provided in the COD documentation). On your TSO terminal (address 0c0), you should see `THIS TERMINAL IS LOGGED ON TO UNFORMATTED SYSTEM SERVICES`.

Move to the TSO terminal session and enter `LOGON DRVUSER`. The password is also `DRVUSER`. This action should produce a CustomPac Master Application Menu. The "P" option should start a normal ISPF session.

To shut down z/OS cleanly, log off from TSO and use the following commands at the MVS console:

```
p tso                                  (Stop TSO.)
z net,quick                            (Stop VTAM.)
p lla
p vlf
   (Wait a few seconds for messages.)
$pjes2,term
   (Wait a few seconds for messages.)
z eod
```

Subsequent IPLs do not need to use CLPA or COLD start with JES2.

12.12.1 TCP/IP connection

We wanted to establish a TCP/IP connection to the COD z/OS system. The COD contains many OSA definitions. We decided to use OSA addresses 0300 - 0302. The profile parameters for TCP/IP are in `TCPIP.SEZAINST`; the VTAMLST parameters are in `SYS1.VTAMLST`; and the TCP/IP procedure is in `SYS1.PROCLIB`. We proceeded as follows to create a TCP/IP tunnel link to our base Linux (which is 10.1.1.1 for the tunnel):

1. Add OSA definitions to our devmap (and restart ISV zPDT):

   ```
   [manager]
   name awsosa 0006 --path=A0 --pathtype=OSD --tunnel_intf=y
   device 0300 osa osa
   device 0301 osa osa
   device 0302 osa osa
   ```

2. Perform an IPL of the COD and start TSO (in a local 3270 session). We examined the default TCP/IP profile in `TCPIP.SEZAINST(PROFILE)` and found it too complex. We created member `PROFILE2` as follows:

   ```
   ARPAGE 20
   AUTOLOG 5
     FTPD JOBNAME FTBD1
   ENDAUTOLOG
   PORT
   ```

```
  7 UDP MISCSERV
    (copy all the PORT statements from PROFILE)
  3389 TCP MSYSLDAP
SACONFIG
;
DEVICE PORTA MPCIPA
LINK ETH1 IPAQENET PORTA
HOME 10.1.1.2 ETH1
BEGINRoutes
ROUTE 10.1.1.0 255.255.255.0   =      ETH1 MTU 1500
ROUTE DEFUALT               10.1.1.1 ETH1 MTU DEFAULTSIZE
ENDRoutes
TCPCONFIG RESTRICTLOWPORTS
UDPCONFIG RESTRICTLOWPORTS
IPCONFIG  NODATAGRAMFWD
START PORTA
```

3. We altered `SYS1.PROCLIB(TCPIP)` so that the **`PROFILE DD`** statement points to member `PROFILE2` instead of member `PROFILE`.
4. We added member `SYS1.VTAMLST(OSATRL1)` as follows:

   ```
   OSATRL1 VBUILD TYPE=TRL
   OSATRLE1 TRLE LNCTL=MPC,READ=(0300),WRITE=(0301),                  X
                 DATAPATH=(0302),PORTNAME=(PORTA),                    X
                 MPCLEVEL=QDIO
   ```

   Be certain that the continuation marks are in column 72.
5. We edited member `SYS1.VTAMLST(ATCCON00)` member as follows:

   ```
   TSOAPPL,TCPAPPL,OSATRL1
   ```

6. We performed another IPL of z/OS. After the start process, we entered z/OS commands as follows:

   ```
   v 300-302,online                    (Bring the OSA ports online.)
   s tcpip
   s tn3270
   ```

We can ping `10.1.1.2` from Linux. Starting a new x3270 session that is connected to `10.1.1.2`, we received an `UNFORMATTED SYSTEMS SERVICES` prompt and can log in to TSO. (We needed to first log off from the local 3270 session because we had only a single TSO user ID.) When shutting down z/OS, you should also stop TN3270 and TCP/IP.

LAN configuration can be frustrating because there are so many details that must be exactly correct. The steps that are described here create a link to the base Linux TCP/IP, which is probably the most basic TCP/IP configuration that is possible with the COD and ISV zPDT. It is a good starting point before attempting to configure an external TCP/IP link.

12.13 WLM and ADCD z/OS

The recent z/OS ADCD systems are provided with a WLM service definition installed. Although there is nothing wrong with the supplied WLM definition, it might not be useful for typical ISV zPDT usage. As a practice exercise, we created another WLM service definition. The following brief description of the process might be useful to a new ISV zPDT owner who has not previously worked with WLM. (The specific parameters that are used in the example are not intended as recommendations.)

The process that is described here is not needed (and probably not appropriate) for basic ISV zPDT and ADCD z/OS users.

WLM service definitions are created and used in the WLM coupling data sets.[26] It is possible, although not required, to extract and write the WLM service definition to a partitioned data set (PDS). A number of PDS members are created when doing this task, and the data is not in a "human" readable format. The same PDS can later be installed in the coupling data set (overwriting what is there) and then activated. At the time of writing, the WLM coupling data set is `SYS1.ADCDPL.WLM.CDS01` (and `CDS02`); the PDS copy is `ADCD.Z24A.WLM`.

WLM definitions can be simple if only simple controls are needed, which is likely to be the case for an ISV zPDT system. A complete WLM definition is a *service definition*. A service definition contains one or more *service policies*. A service policy involves workloads, service classes, and classification rules. A simple WLM definition might contain only these elements. Only one service definition can be installed for a WLM at any one time[27] and only one *policy* within that service definition can be active at any one time. However, you can work on an additional service definition (by using ISPF) in the coupling data set while a different service definition is being used.

We designed a service definition that is named BASEAD that has a single policy that is named BASEAD1, with the characteristics that are shown in Table 12-1.

Table 12-1 Simple WLM definition

Workload name	Service class name	Importance	Service class goal	Classification
HIGH	FAST	1	90% velocity	LDAP, LSFM, STC, and TCP
TSO	TSO	2	90% in 0.25 seconds	TSO
CICS	CICS	3	80% in 1 second	EWLM, CICS, and IMS
SERVERS	MEDIUM	3	50% velocity	CB, ASCH, Db2, DDF, IWEB, IBM MQ, and SOM
BATCH	SLOW	4	Discretionary	JES

The workload names and service class names are arbitrary. With a more complex definition, there are common naming conventions, but we used our own names for this simple example. If multiple service classes are not meeting their goals, then WLM adjustments are made in the order of importance (with 1 being the most important). The "90% velocity" indicates that work in this service class is expected to run at 90% of the processor speed. The "90% in 0.25 seconds" indicates that 90% of the transactions in this class should complete in 0.25 seconds or less. Discretionary means that there is no specific goal and usually indicates the lowest performance class. (In some cases, the discretionary goal can lead to performance degradation if the associated jobs use zIIP functions.)

[26] Coupling data sets are normally associated with sysplex operation, but the WLM usage is independent of whether a sysplex is being used.

[27] Only one service definition can be installed in the WLM coupling data set at any one time. The key word is "installed", which means a policy within it can be activated. A different service definition can be concurrently built (by using ISPF) in the same coupling data set, but it is not "installed" now.

Our service definition is a little odd in that it places TSO importance above other servers, such as Db2, CICS, IBM Information Management System (IMS), and IBM MQ. This unusual definition suited us because we use TSO to monitor other work and wanted our monitoring activity to have priority over most other work. The CICS and SERVERS definitions are separate only because WLM rules prevent assigning a velocity goal to CICS, IMS, and EWLM, but if this situation were not the case, we would have placed all of these definitions in the SERVERS workload.

There are many IBM manuals, IBM Redbooks publications, and third-party documents that describe WLM parameters in great detail, so we do not attempt to repeat any of this material. The purpose of the description here is to describe the mechanics of creating and using your own WLM service definition with our trivial example. As with most z/OS administrative activities, there are many ways to go about this task, and we describe a basic approach that can be used as a starting point for more complex work.

Proceed as follows (by using a recent z/OS ADCD system):

1. Decide on the basic design of your service definition, including the initial workload names and service class names. You can expand them later.
2. Create a small PDS to save your workload after you define it. Each service definition requires a separate PDS for saving it. As a best practice, use five tracks, five directory blocks, LRECL 80, and RECFM Fixed Block (FB).
3. Go to ISPF option M.12 to start a WLM interactive dialog. Select option 3 to create a definition. (This best practice depends on the current ADCD z/OS setup.)
4. On the next panel, enter a Service Definition name of your choice and a brief description. Then, select option 1 to specify a policy name of your choice. (Press PF3 twice to return to the main WLM menu.)
5. Select option 2 to create a workload name. (Press PF3 twice to return to the main menu.) (Within the workload panel, use option 1 to create another name.)
6. After defining your initial workload names, use option 4 to define service classes. For each one, assign a service class name and the name of the corresponding workload that you previously created. Then, position the cursor on the Action field and enter I (for `Insert new period`). This action opens a small panel where you can select a goal for this service class, such as a velocity percentage, and an importance parameter and duration (to be used if there are multiple periods for this service class).[28] If you entered a duration parameter, then you should insert another period specification. The last (or only) period for a service class does not have a duration. You should assign a service class for each of your workload names.
7. After defining your workloads and service classes, use option 6 for `Classification Rules`. This action opens a panel with about 17 predefined classifications. Do not attempt to create classifications unless you understand how the whole z/OS system relates to WLM. For each line in the panel, select option `3` (`Modify`). In the following panel, go to the Service `DEFAULTS` field and enter the name of one of your service classes, and then press PF3. If you do not associate one of your service classes with each of the classifications, it defaults to a `SYSTEM` level service class, which is an automatically provided high-level service class.
8. When you have assigned all the classifications, press PF3 to go the main menu and select the `Utility` option at the top of the panel. You can validate your new definitions by using one of the options. (The validate function appears to check syntax, but makes no attempt to do more than that.)

[28] There is a huge body of documentation about detailed WLM definitions. You can search this material for information about goals, durations, and other groupings.

9. Press PF3 to leave the WLM dialog. You should see a panel where you can save your definition to a data set. Use this option to save the definition to the PDS that you defined for it. Optionally, you can use option 3 to discard all your work. This option can be useful if you are experimenting with the WLM panels.

There are many other choices in the WLM dialog panels, but the options that are described are sufficient to create a basic WLM service definition.

If you followed these steps, you now have your service definition that is saved in a PDS. To use it, start the WLM dialog again (`M.12`) and select option 1 to read a saved definition. The resulting panel might have the name you that used to save your definition. If you enter a question mark in this field, you should see the names of other saved WLM service definitions, which are likely to include `ADCD.Z24A.WLM` or a similar name. Complete the following steps:

1. Select one of the saved names, such as your own data set or the ADCD data set if you want to revert to the default ADCD WLM service definition.
2. In the main WLM dialog panel, select the `Utilities` option at the top line of the panel, and select option 1 to install the service definition. You see a subpanel asking permission to overwrite whatever service definition is installed. Reply `YES` to this question.
3. If you use the **`D WLM`** operator command, you see that the previous service policy is still being used. In the WLM dialog, select `Utilities` again, and select option `3` (`Activate Service Policy`). Select a service policy from the list that is presented, which activates the service policy. To confirm the service policy, run **`D WLM`**.

Is there any point to building your own WLM service definition when working with an ADCD system on ISV zPDT? There is no simple answer. WLM is most effective in a system running sustained mixed workloads. For practical purposes, most WLM actions result in changing z/OS dispatching priorities, which is most meaningful if the system has a sustained workload approaching 100% CPU utilization. Adjustments are made at something like 10-second intervals. An ISV zPDT system with a few TSO users editing, compiling, and unit-testing modules presents an erratic workload that does not respond to WLM well. However, a heavy test workload that involves batch, Db2, and extensive (simulated) terminal interaction might respond to WLM well.

12.14 Telnet

With z/OS ADCD releases, you can connect to TCP/IP port 1023 with a simple **`telnet`** command connection. (This situation might change or be deleted with future ADCD z/OS releases.) There are many PC-base equivalents to the old **`telnet`** command (which might not be available). A common equivalent of **`telnet`** is the **`putty`** command that is used in telnet mode. (Any form of **`telnet`** is insecure, but there might be a special need for it.)

The 1023 port might not be configured for some releases. If your connection fails, edit `TCPIP.ETC.SERVICES` and insert the following information into the appropriate place:

```
telnet  23/tcp
otelnet 1023/tcp
```

Some z/OS ADCD releases might not automatically start **`inetd`**, which is probably needed for **`telnet`**. To start it, enter **`S INETD`** on the z/OS operator console. (This function is not required for using SSH.) You might want to add the **`S INETD`** command to the VTAMxx members in your parmlib if the command is not there.

12.15 Disabled waits

You might sometimes see a message in your ISV zPDT Linux window such as this example:

```
Warning! Disabled Wait CPU 0
$ d psw                                 <--You can issue this ISV zPDT command.
PSW for CPU 0  000A0000 00000xxx                          (24/32 bit mode)
PSW FOR cpu 0  00020000 00000000 00000000 00000XXX      (64-bit mode)
```

When you receive such a message, you can issue a command to display the Program Status Word (PSW). The last 3 characters of the PSW should contain a wait state code, and an extended code might also be present in other characters of the PSW. Some wait states are restartable.

The following list covers only a few of the wait state codes, mostly to generally illustrate the basic concepts:

002	During an IPL, an I/O operation was not initiated.
003	The IPL cannot continue because a subchannel is not operational for IPL or the IODF device.
004	During initialization, I/O was not initiated.
005	I/O interruption during the IPL and unit check.
006	I/O error during IPL processing for either the SYSRES or IODF volume.
007	During initialization, the console was not available.
009	System build error (a rare z/OS problem).
00A	Cannot find SYS1.LINKLIB or SYS1.CSSLIB in the catalog.
00B	Master scheduler abended.
00D	Master scheduler abended.
00E	Problem on an SYSRES volume (SYS1.NUCLEUS).
00F	IPL volume does not contain IPL text.
013	Error during NIP.
014	Recursive program checks.
017	Unit check during IPL.
019	IPL program in error (6). Also seen if you perform an IPL on an older operating system that does not support CZAM on a IBM z14 operating in normal CZAM mode.
01B	SLIP requests wait.
040	ABEND during NIP.
044	Machine check during NIP. (Try performing an IPL again.)
064	NIP error and RTM not initialized.
088	IPL: error in LOADxx or NUCLSTxx.
CCC	Wait state generated by **QUISECE** command.
FFx	Non IBM program created a wait state.

13

Additional ISV IBM Z Program Development Tool notes

This chapter contains various notes about Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT). This information is not required for basic ISV zPDT operation, but it is helpful for better understanding ISV zPDT and for more advanced uses.

13.1 Minor ISV zPDT notes

The notes that are shown here were relevant at the time of writing, but personal computer (PC) Linux is known to have frequent minor changes or updates, so some of these notes might not apply to later versions:

- Known z/VM issue when z/VM hibernates

  If an ISV zPDT base PC is running z/VM and the PC is placed in a hibernation state (by closing the laptop lid, for example) for a lengthy period, then z/VM might fail with a `PRG009 ABEND` message. In this example, the "lengthy period" is 17 hours. The underlying problem relates to time-of-day (TOD) clock skews on the PC, and there are multiple variables that are involved. This situation is rare, and at the time of writing is a permanent restriction for ISV zPDT.

- Linux library prune

  It is possible to ""prune" a Linux system to remove unused libraries. Unfortunately, this action sometimes removes the 32-bit libraries that are needed by the SafeNet modules that are used by ISV zPDT. If you must "prune" your Linux system, you might need to reinstall the 32-bit library that is needed by SafeNet. The name of the library varies with different distributions, as described in 5.1, "Linux installation" on page 126.

- Unexpected termination

  In some cases with Ubuntu (more recent releases), the ISV zPDT session might terminate under some conditions, especially if the terminal window that is used to start ISV zPDT is closed. The following command resolved this situation in one case:

  ```
  sudo loginctl enable-linger
  ```

  The Linux "improvement" behind this condition might migrate to other Linux distributions or might disappear. It was unexpected when we encountered it.

- S/390 Compatibility Mode

  S/390 Compatibility Mode is available in IBM z15 (and later) systems and in ISV zPDT GA10 (and later). However, some functions within this mode are "model-dependent" and might react differently with an ISV zPDT "model" than with a "real" z15 or later machine. S/390 Compatibility Mode is used by the Conversational Monitor System (CMS) (running under z/VM), and is used internally by a few components of other operating systems.

  Some instructions (or suboptions of certain complex instructions) that do not exist in an S/390 system might function in S/390 Compatibility Mode. Proper S/390 programming would not attempt to use such instructions (or options) because they do not exist on an S/390 system.

- Free IBM zSystems Integrated Information Processor (zIIP) processors

  Starting with ISV zPDT GA8, a token license is not needed for emulated zIIP processors. However, the "free zIIPs" count toward the maximum of eight emulated processors per ISV zPDT instance and the requirement for the number of PC cores to be at least equal to the number of emulated processors. Also, there cannot be more zIIPs than CPs.

  You should be aware that zIIP usage by z/OS interacts with workload manager (WLM) options (and parmlib options) and in some cases can substantially reduce your system performance. If a workload is in a discretionary dispatching level and zIIP-eligible work is dispatched, then the task is dispatched only on a zIIP, even if idle CPs are available.

For example, a user with a 1090-L03 token (three licenses) that uses a PC with four cores can configure three CPs and one zIIP, which creates an ISV zPDT system with the number of CPs equal to the number of cores. Will this configuration improve performance? The usage of “free” zIIPs might appear attractive to all z/OS users (assuming that they have sufficient cores in their PC.) However, as a best practice, doing some performance testing might be a good idea because the zIIPs might add performance or, in a few cases, might harm performance.

With ISV zPDT releases GA11 and later, the “free zIIPs” are no longer a “new” offering but the situation is still confusing to some users.

- zPDT build information

  The `/usr/z1090/bin/librarybuild` file contains information about the levels of Linux that are used to create the copy of zPDT. In general, you should not use a version of Linux that is earlier than the libraries that are noted in this file.

- Copying emulated direct access storage device volumes

  Never use Linux to copy an emulated direct access storage device volume while ISV zPDT is active and potentially using that volume, that is, if the emulated volume is in the device map (devmap). For example, *never* use Linux **`cp`**, **`rsync`**, **`gzip`**, or similar commands to copy a potentially active volume. The Linux commands are unaware of any buffer or cache information that is held within z/OS that must be written to the volume. Violating this rule can lead to corrupted data or a corrupted emulated volume format.

  When ISV zPDT is not active, any Linux commands that copy files can be used to copy (“back up”) an emulated volume. An emulated direct access storage device volume is a single Linux file. As a practical matter, the **`gzip`** command is often used to copy and compress a volume.

- Token dates and times

  The ISV zPDT tokens remember the latest date and time that they obtain from the underlying Linux system. If the token sees the date or time move backward, a *time cheat* condition is produced, and ISV zPDT does not start.

  For example, if you set the PC date ahead several months (perhaps to test an expiration function in the application that you are developing) and you use ISV zPDT with this advanced date, you cannot then return to the correct date and use the same token. If you inadvertently used an incorrect (future) date with the token and you now find the token unusable with the correct date, you should contact your ISV zPDT supplier.

  If you *must* temporarily change the PC clock (when not using ISV zPDT), remove the token before changing the clock and reset the clock to the current time before adding the token again. If you move a token among multiple PCs, ensure that the hardware time-of-day (TOD) clocks reflect times close to each other on all the machines.[1]

  The **`settod`** command that is provided with ISV zPDT provides a way to test software by using different dates, but it does not change the value of the PC hardware clock or the Linux software clock.

- Typing OPRMSG too many times.

  The default IBM zSystems console (normally a Hardware Management Console (HMC)) is emulated by ISV zPDT by using the Linux window that was used to start ISV zPDT. Operating system output that is sent to the default console appears in the Linux window. To enter IBM zSystems commands from the default console (that is, from the Linux window), use the **`oprmsg`** command, as in this example:

  ```
  $ oprmsg 'CN(*),ACTIVATE'
  ```

[1] This statement is a technical one. Your zPDT license agreement might restrict this usage.

Typing **oprmsg** for every input line becomes tedious. The Linux **alias** function can be used to assign a single character to create the **oprmsg** text:

```
$ alias +=oprmsg              (Use th plus character to create oprmsg.)
$ + 'cn(*),activate'
$ + d a,l
```

A space is needed after the plus sign (+) like a space is needed after the **oprmsg** command before the command text is entered. Single quotation marks might be needed to prevent the Linux shell from processing special characters such as parentheses.

- PC Hyper-Threading

  As a best practice, PC Hyper-Threading should be disabled in the BIOS of any PC running ISV zPDT. Sometimes, you encounter z/OS "excessive spinloop" messages when running with Hyper-Threading enabled, and disabling Hyper-Threading usually eliminates these messages. While we cannot verify our assumption, we assume that z/OS is spinning on one "half" of a core, and the z/OS process that would resolve the spin is waiting for cycles on the other "half" of the same core.

  With limited Hyper-Threading experimentation, we have not seen spinloop messages recently while running simple batch and Time Sharing Option (TSO) sessions. However, this absence does not mean that the spinloop situation does not exist, but means that any spinloop conflict does not last long enough to trigger a z/OS message.

  All formal zPDT testing by IBM occurs with Hyper-Threading disabled. This approach is the "safe" way to run zPDT. If you have a larger server with many cores, there is probably no need to consider Hyper-Threading. If you have a common environment with an L03 token and a four-core PC and verifying zIIP processing is important to you, you might consider the experience that we related here. However, some ISV zPDT users operate with Hyper-Threading enabled and seem to experience no problems.

  Important: We have not experimented with the C/C++ compiler SMP option. This option provides for parallelization of a program, which might create more exposure for spinloop problems when using Hyper-Threading.

  We also saw problems that involved heavy z/OS Java loads (such as z/OSMF) when using Hyper-Threading.

- Important Linux command-line interface (CLI)

  The Linux CLI that is used to enter the **awsstart** command is important. Asynchronous messages from ISV zPDT are sent to this window. (User commands that are sent to ISV zPDT can be entered from any Linux window that is operating under the same user ID that started ISV zPDT). Asynchronous messages include ISV zPDT error messages (such as an unexpected channel-to-channel (CTC) disconnection) and z/OS messages that are directed to the HMC console.

  If you inadvertently close the Linux CLI that is used to start ISV zPDT, you do not see these asynchronous messages. At the time of writing, there is no way to recover the function of this panel.

- Linux “out of memory”

  In rare circumstances, you see an “out of memory” error message from Linux, typically when starting ISV zPDT. Assuming that your ISV zPDT runs correctly most of the time, we believe that this message is related to fragmentation of the shared memory locations in Linux. ISV zPDT uses Linux shared memory extensively, and ISV zPDT startup, shutdown, configuration change, startup, shutdown, and so forth might eventually result in an out-of-memory message. Advanced Linux users might try to rework shared memory parameters (while ISV zPDT is not running), but the easiest solution is to restart Linux. We emphasize that this condition is rare.

- The crontab and sudo entries

  ISV zPDT places entries in the Linux cron tables at root level. You can see them as follows:

  ```
  $ su                            (Change to root.)
  # crontab -l                    (List crontab entries for root.)
    @reboot   /usr/z1090/bin/safenet_daemons_restart  reboot > /dev/null
    */11 * * * * /usr/z1090/bin/safenet_daemons_restart > /dev/null
  # exit                          (Leave root.)
  ```

  Remote license servers and Unique Identity Manager (UIM) functions result in extra cron entries. Do not change or delete these entries if you use cron functions for other purposes. If you use the **`SecureUpdate_authority`** command, an entry is added to the Linux `/etc/sudo` file for every user ID that you authorize. Do not remove these entries from `/etc/sudo`.

- Linux disk partition full

  In rare cases, you might find a Linux disk partition unexpectedly full, but normal Linux commands do not list any files that would fill the partition, and deleting files in the partition does not seem to help. (This issue is not unique to ISV zPDT, but it is something that can be frustrating to ISV zPDT users.) The following Linux commands might be useful:

  ```
  su -           (Switch to root to display more information.)
  cd xxxxx       (Change to the relevant directory or partition.)
  ls -al         (List the file names and sizes.)
  ```

  You see no files with excessive sizes:

  ```
  df -h          (List sizes and the “fullness” of all the partitions.)
  ```

  Your partition seems full:

  ```
  du -h          (Read the instructions for this command.)
  ls -al .       (Notice the period to help list “invisible” files.)
  dmesg          (Does these files seem large?)
  rm .xxxx       (Delete unreasonable files by using a period before the names.)
  ```

  Check your partition again to see whether these commands corrected the problem.

  If you do not find the full partition, try switching to different file systems or directories and running the same set of commands again. In the specific cases that we saw, the excessively large file (when it was finally found) seemed to be in the same format as typical **`dmesg`** records. (The “normal” **`dmesg`** files are typically in the `/var/logs` directory, but the excessively large files we found were in other locations.)

  There are multiple ways that a disk partition might appear full, and the notes here are not inclusive.

- Too many directory levels

  While not especially related to ISV zPDT, some Linux tools seem to create many directory levels, making it tedious to delete all the files that are involved because the common **rmdir** command cannot delete a directory that contains files. Change to the directory that owns the top level of the offending structure and then issue the command **rm -rf *xxxx*** (where xxxx is the offending directory name). This command deletes the named directory and anything that it holds (including subdirectories). Be careful with this command.

- Copying large Linux files

  While not a specific ISV zPDT issue, you should be aware of a particular limitation in copying Linux files larger than about 4.4 GB. (We encountered this limitation when one of the z/OS Application Development Controlled Distribution (ADCD) `gz` files was over this size.) Most USB flash drives apparently come with FAT32 file systems, which places a limitation on the maximum file size that can be handled. In our case, there was a PC panel message (that we sometimes did not notice) about a *splice* error because the file was too large. In a more obscure version of the problem, the same error occurred when attempting to copy the file to a DVD. The solution (for a USB flash drive) is to change the file system format to ext4 or one of the other recent Linux file system types.[2] (The error does not occur when copying between two Linux file systems that are not in FAT32 format.)

- Doing clever things with Linux disk partitions

  Although not a direct ISV zPDT issue, we noted some obscure customer problems that seem to occur after manipulating disk partitions on disks that are shared among multiple Linux systems.[3] The problems seemed to be related to extended partitions for the basic Linux directories. We have not seen reports of this situation from many ISV zPDT customers, but have seen it from those customers who use the same PC or disks for a dozen (or more) other operating systems, projects, configurations, and so forth.

- Odd messages about direct access storage device during initial program load (IPL)

  During z/OS IPL, a number of console messages beginning with the text "`CU AUTHORIZATION FAILED...`" might be seen. These messages should be ignored because they do not indicate a real problem with the emulated direct access storage device volumes.

13.2 Need for Linux root operation for Open Systems Adapter

Earlier releases of zPDT required Linux root access for some functions, especially for the Open Systems Adapter (OSA) emulation function. Starting with fixes for ISV zPDT GA10, root access is no longer required.

The detailed operation of the earlier releases required a setuid permission bit for module eDMosa, which is no longer required. Instead of the setuid bit (to obtain root access), zPDT uses the Linux **cap** or **setcap** function, which should not be disabled. You can check whether these functions are enabled by running the following command:

```
$ getcap /usr/z1090/bin/eDMosa
/usr/z1090/bin/eDMosa = cap_net_admin,cap_net_raw+eip     (expected result)
```

The result that is shown here, at the time of writing, was based on CentOS, so other or later Linux versions might be different.

[2] We assume that changing a USB flash drive file system format results in the deletion of whatever files are on the USB flash drive. Be careful.

[3] In one instance, the problem appeared to be local area network (LAN) operations that worked in some instances but failed (for no apparent reason) in other instances.

13.3 The cpuopt statement

The **cpuopt** statement in a devmap specifies optional parameters for the CPs. At the time of writing, here are the valid parameters:

cpuopt asn_lx_reuse=on	No blanks in the operand
cpuopt asn_lx_reuse=off	No blanks in the operand
cpuopt zVM_CouplingFacility	No blanks in the operand
cpuopt alr=on,zVM_Coupling	Abbreviations
cpuopt ZARCH_ONLY=NO	Uppercase with no spaces)

The **asn_lx_reuse** operand can be abbreviated as **alr**. The **zVM_CouplingFacility** operand can be abbreviated as **zVM_CouplingFac** or **zVM_Coupling**. These operands do not contain blanks, so be certain that no blank exists before or after the equal sign.

The **ZARCH_ONLY** parameter is new with zPDT GA8. It defaults to `YES` and is the standard mode of operation for IBM z14 systems. It causes the IBM z14 to undergo an IPL in normal z/Architecture mode. Specifying **ZARCH_ONLY=NO**[4] causes zPDT to undergo an IPL in ESA390 mode, but the machine otherwise operates as a IBM z14 system. This configuration is a non-standard one that might be useful with older operating systems that are not designed to undergo an IPL in z/Architecture mode. Using this option creates an environment that is not supported by IBM, and it should be used with caution. (The **ZARCH_ONLY** function is sometimes documented as the CZAM facility.) Performing an IPL of an older z/OS operating system (that does not support IPL in z/Architecture mode) on a IBM z14 can produce disabled wait state 19. The **ZARCH_ONLY=NO** option might be useful in such situations. The default is **ZARCH_ONLY=YES**.

The **asn_lx_reuse** parameter defaults to "`on`", which 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 might be useful for running early z/OS releases. The usage of `alr=off` produces an environment that is not supported or tested by IBM. Although this model might be useful for working with earlier z/OS releases, the user must assume all responsibility for the correctness of operation and results.

The **zVM_CouplingFacility** operand is significant only for IBM Z Development and Test Environment (ZD&T) systems, which must have the proper license feature to enable it. In effect, the **zVM_CouplingFacility** function is always present for ISV zPDT systems.

13.4 Read-only and shared direct access storage device

Emulated direct access storage device volumes can be used as read-only volumes by setting the Linux permissions to disallow writing to the Linux file that contains the emulated volume. For example, assuming that you have a 3390 volume that is stored in `/z/WORK02`, the following Linux command[5] makes it read-only for the owner (normally, the Linux user ID that started ISV zPDT), the owning group, and all other user IDs:

```
$ chmod 444 /z/WORK02        (You can use other forms of chmod.)
```

[4] The **ZARCH_ONLY=NO** option also disables the CM390 feature of the IBM z14 architecture, which should have no direct effect for most users.

[5] Depending on your Linux file ownership, you might need to operate as root to issue this command.

The execute permission for the file is not relevant. A more detailed permissions setting might be used, but the result should be to prohibit write permission to ISV zPDT. An alternative method is available as an option with the **`awsmount`** command.

Important: Sharing, as controlled by Linux or zPDT, is not the same as "shared direct access storage device" that is controlled by z/OS.

Older z/OS levels

When z/OS accesses the volume, an error message might be displayed (probably due to an attempt to update the Volume Table of Contents (VTOC) statistics), but the operation continues in read-only mode.[6] You can browse data sets, for example. If you attempt to change a data set (with IBM Interactive System Productivity Facility (ISPF), for example), an error message is produced, and you must cancel the SAVE operation. The error messages are like the following example:

- On the MVS console: `IOS000I 0AA0,01,WRI,1D.........`
- On the MVS console and TSO: `IEC212I 414-04,IFG0201......`

We informally used basic sequential data sets, partitioned data sets (PDSs), and PDS/E data sets without problems. We were unable to use Virtual Storage Access Method (VSAM) data sets on a read-only volume, but there might be techniques to bypass this restriction. z/OS APAR OA50068 provides z/OS support for read-only volumes and extended documentation for their use. ISV zPDT implementation of read-only (at the Linux permissions level) differs from the technique that is described in the APAR, which might produce operational differences.

Recent z/OS levels

z/OS recently added more formal support for read-only volumes, which is documented in *z/OS DFSMS Using Data Sets*, SC23-6855. With this support, JCL support and a degree of VSAM support are available for read-only volumes.

13.4.1 Shared read-only volumes

You can share read-only direct access storage device volumes. The sharing is done at the Linux level and not visible to z/OS. You do *not* use the **`--shared`** option with the awsckd entry in your devmap. Because of the read-only nature of the volumes, there is no need to coordinate Linux disk cache operations. Various Linux facilities are available to share files. The most common is NFS.

Important: The usage that is described here requires all access to the volume to be read-only. It is not suitable, for example, to allow one ISV zPDT system to have write access while all other sharing systems are read-only.

[6] We noticed that z/OS 2.1 is less likely to produce error messages than earlier z/OS releases.

There are many ways that you might configure such an operation. The more obvious example involves sharing a direct access storage device among different zPDT instances on the same PC. A more complex example might involve NFS sharing, as follows:

1. We placed the intended read-only 3390 volumes (that is, the Linux files containing these volumes) in a separate directory (named `/z3`) on our first Linux system and changed the permissions on each file in the directory to read-only.
2. We started an NFS server on this Linux system and indicated that we wanted to export `/z3` as read-only mode (specified as `ro`). We did not use NFSv4 or GSS security. Depending on your firewall status, you might need to open a port in your firewall. You can limit your export operation to specific clients or export to anyone. Other Linux systems (and more recent versions) have various methods of configuring NFS server operation, and you must use whatever interface is appropriate for your Linux version.
3. We changed our devmap to find the read-only volumes in their new directory (`/z3`).
4. On our second zPDT machine, we defined a mount point. In our case, we named it `/z3` to match what we did on our first Linux system, but any mount point name can be used.
5. We configured an NFS client on this machine, mounting `/z3` (from our server) to `/z3` (on our local client).
6. On the client, we changed to root and issued a **`mount`** command:

   ```
   # mount -t nfs 192.168.1.80:/  /z3
   ```

   Our NFS server used IP address `192.168.1.80`. We needed `192.168.1.80:/` and not `192.168.1.80:/z3`[7] in the **`mount`** command. In principle, we should be able to see the shared volumes by issuing the **`ls /z3`** command on the client machine. In practice, we sometimes must restart the client to have the contents of the remote `/z3` directory appear on our client `/z3` mount point. The following line appears in `/etc/fstab`:

   ```
   192.168.1.80:/z3 /z3 nfs defaults 0 0    (Assuming 192.168.1.80 is the server.)
   ```

7. We changed the devmap for our second zPDT to point to the read-only volumes at their location in `/z3`.
8. We start ISV zPDT and perform an IPL of z/OS on the server and client. We were able to access the shared volumes (read-only) from both systems.
9. If you want to change the contents of a read-only volume, you must disable the client systems so that no “stale” information is available for volumes. In practice, this action means ending ISV zPDT operation and possibly closing down Linux. (We did not explore the exact details of this scenario.) You end ISV zPDT on the server; change the permissions for the volumes that you want to alter; start ISV zPDT; perform an IPL of z/OS; make the changes; end ISV zPDT; and change the file permissions back to read-only. Then, you can restart the client system (or systems) and ensure that they can “see” the volumes at the mount point. This scenario is not convenient, and read-only operation is not appropriate for volumes that are frequently updated.

   One reason for this seemingly excessive technique is that residual data for the volume might be in various forms of z/OS caches, or in the base Linux disk cache.

We noticed that ISV zPDT start on the client was a little slower than with the previous operation and z/OS shutdown (with a **`quiesce`** command) was definitely slower. The details in these notes might change with more recent PC Linux releases, but the overall technique should be about the same.

[7] The **`mount -t nfs 192.168.1.80:/z3 /z3`** command did not work.

13.5 zPDT log files

zPDT writes a number of log files that are based in the ISV zPDT user's home directory. If Linux user `ibmsys1` starts ISV zPDT, the log files are in `/home/ibmsys1/z1090/logs`. Most of the log types are for IBM use when working with a problem and are not interesting otherwise. The log types include the following ones:

`awsstart_`*`pnnnn`*`_000001.log`	One line. Created when starting zPDT.
`CPU_QUEUE_SEQMEWNT....gz`	For internal use only.
`dv_0A8A_`*`pnnnn`*`_000001.log`	Created when starting each device.
`emily_uim_`*`pnnnn`*`_000001.log`	Created when starting a UIM server.
`log_console-`*`pnnnn`*`-t2018-11-03_10.12.00.txt`	Console log.
`summaryFile`	IBM debugging only.
`tnport12_`*`pnnnn`*`.log`	Created by aws3274.
`uimclient_`*`pnnnn`*`_000001.log`	Created by the UIM client start.

The “pnnnn” in these descriptions is the Linux process ID (PID) that was used. Older log files are not immediately cleared and typically have various PID numbers.

One of the log types might be more interesting to ISV zPDT users, which are the `log_console` files. There might be several of these files. The name includes a timestamp indicating when the log file started. In our example, the `log_console` was started November 3, 2018 at 10:12 AM. The `log_console` file includes the following information:

- Start messages from ISV zPDT.
- Asynchronous messages from ISV zPDT that are displayed in the Linux command terminal that is used to start ISV zPDT.
- ISV zPDT commands that are entered in any Linux terminal window.[8]
- Messages from z/OS (or another operating system) that are directed to the “HMC hardware console.” These messages are displayed in the Linux command terminal that is used to start ISV zPDT.

Notice the two categories of messages that are displayed in the Linux command terminal that is used to start ISV zPDT. If this Linux command terminal is closed, these messages are not displayed anywhere but are still written to `log_console`.

The following Linux commands might be useful to zPDT users:

```
oprmsg ‘V CN(*),ACTIVATE’
oprmsg d a,l                    (A z/OS command)
```

These commands activate the “HMC hardware console” as a z/OS operator terminal. The **`oprmsg`** commands can be entered in any Linux CLI, but the resulting messages from z/OS are shown only in the Linux window that was used to start ISV zPDT and are also placed in the `log_console` file. If you want to automate (from Linux) the various z/OS commands and responses, these commands might be useful.

[8] Any Linux terminal window operating under the Linux user ID that started zPDT. This detail is relevant if multiple zPDT instances are operational.

The optional **oprmsg_file** statement in the ISV zPDT devmap is used for direct placement of the `log_console` file in a different location. For example, the following stanza writes the log to `/tmp/sysmessages` instead of a file in the `/home/ibmsys1/z1090/logs/` directory:

```
:[system]
memory 20G
processors 3
oprmsg_file /tmp/sysmessages
...
```

Numerous logs can accumulate in the default `logs` directory. You can clear them by using the **--clean** option with the **awsstart** command. A fix pack for ISV zPDT GA10 (and for later releases) includes changes to reduce potentially the amount of log information that is written.

13.6 Large PC memory

Large PC memory usually improves ISV zPDT performance. Large memory helps you avoid paging and provides a large Linux disk cache function. In rare cases, large PC memory[9] can create a problem. A rare problem situation might be as follows:

- The z/OS workload is not using especially large memory, for example, Db2 with active large tables is not being used.
- The z/OS system is quickly producing many multiple asynchronous disk operations (especially write operations).
- RAID 5 usage might be involved. RAID 5[10] requires two disk writes for each logical write operation.
- The PC memory is large enough that the Linux disk cache can absorb many gigabytes of disk output without writing to disk.
- Linux decides that it has too many “dirty” pages in the cache and must write them to disk before allowing more disk operations.[11]
- z/OS starts seeing disk timeouts, typically as Missing Interrupt Handler (MIH) actions, and can create many error messages. z/OS usually eventually resolves the situation.

We have seen this situation when using pathological test cases. We have rarely seen it in normal usage. One extreme example that we investigated was as follows:

- The PC server had 160 GB of memory.
- z/OS was running under z/VM.
- RAID 5 was being used.
- The user defined five new, full 3390-9 volumes for Job Entry Subsystem (JES) 2 spool space.
- When cold starting JES2 the first time, it started formatting the five volumes, asynchronously, in parallel. (JES2 formatting is fast and efficient, which contributed to the overloading of the Linux I/O functions.)
- A short time later, the MVS console was flooded with MIH and other error messages. The system did not recover in this situation.

[9] In this case, “large” means *at least* 100 GB.
[10] Most RAID adapters have sizable cache memory. This discussion assumes volumes of data such that the RAID adapter cache is relatively small.
[11] This description is probably an inaccurate one of exactly how Linux manages the disk cache, but it describes the general situation.

In this particular case, running the same z/OS functions without using z/VM allowed the JES2 formatting to complete. (There were still MIH and other disk error messages, but the functions completed.) Subsequent usage of the z/OS system under z/VM operated normally.

Recent Linux levels appear to better handle situations like the one that is described here. We stress that the overrun situations that are described here are rare. We often use PCs with large memory (256 GB, most recently) with no problems during "normal" z/OS operation.

13.7 Dynamic configuration changes

ISV zPDT does not support dynamic changes to an operational devmap. The selected devmap is read when ISV zPDT is started (by using the **awsstart** command) and any changes to the devmap after that point are not effective unless ISV zPDT is stopped and restarted.

The most common reason for wanting to change the devmap is to alter the direct access storage device configuration, usually by adding more volumes. This task can be done with a little planning. The basic requirement is to include "spare" direct access storage device devices in your devmap by using device numbers (addresses) that are valid for your input/output definition file (IODF). Here is an example:

```
[manager]
name awsckd ABCD
device 0A80 3390 3990 /z/H1RES1        (normal emulated direct access storage
device definitions)
..... (other direct access storage device definitions)
device 0AB0 3390 3990                  (spare device definition)
device 0AB1 3390 3990                  (spare device definition)
```

You can use a spare device to add a work volume while ISV zPDT is running:

```
$ alcckd /z/LOCAL4 -s500 -d3390        (Create emulated volume of 500 cyls.)
$ awsmount 0AB0 -m /z/LOCAL4           (Mount on spare device.)
```

The new volume has no label or VTOC and cannot be varied online to z/OS. You must run an ICKDSF job:

```
//INITVOL  JOB  1,OGDEN,MSGCLASS=X
//         EXEC PGM=ICKDSF,REGION=40M
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
 INIT UNIT(AB0) NOVALIDATE NVFY VOLID(LOCAL4) -
 PURGE VTOC(2,1,15)
/*
```

This job should request the operator to reply `U` to allow the volume initialization. After the ICKDSF job completes, the operator can issue the **VARY AB0,ONLINE** command and the volume is ready for use. You probably want to change your devmap so that the volume is routinely available when ISV zPDT is started the next time.

Intense Linux I/O: The **alcckd** command, when creating an emulation volume, creates intense Linux I/O while it is formatting the volume, which might disrupt or slow concurrent z/OS operation, especially if z/OS jobs are running.

13.8 Security exposures

Although most ISV zPDT usage occurs in environments where base Linux security is not a concern, the following items might be of concern to some users.

13.8.1 Reducing root usage

After ISV zPDT is installed, few functions normally require running as root. The most common are the commands that are involved in updating the token licenses and the **`clientconfig`** command. You can avoid using root in this situation by completing the following steps:

1. Select a user ID (not root) of someone who is allowed to use the token update commands. These commands are **`SecureUpdateUtility`** (for zPDT releases before GA5), **`Z1090_token_update`**, and **`Z1091_token_update`**.
2. As root (probably when installing ISV zPDT), run the following command:

   ```
   # SecureUpdate_authority -a <userid>        (Specify your selected user ID.)
     or
   # /usr/z1090/bin/SecureUpdate_authority -a <userid>
   ```

   The full path name for the command is needed if you are not in the `/usr/z1090/bin` directory when issuing the command. This command is issued only once, and it makes an entry in the `/etc/sudo` file. To remove a user ID from the authorized list, run the following command:

   ```
   # SecureUpdate_authority -d <userid>
   ```

   or

   ```
   # /usr/z1090/bin/SecureUpdate_authority -d <userid>
   ```
3. Thereafter, use the **`sudo zpdtSecureUpdate`** command[12] while operating as the selected user ID. This command automatically switches to the `/usr/z1090/bin` directory and runs the appropriate commands with root authority. The **`sudo`** function provides temporary root authority, and the **`zpdtSecureUpdate`** command temporarily switches to the required directory, runs the appropriate token administration command, and returns to the previous directory.
4. Select a user ID (not root) who is allowed to use the **`clientconfig`** command and issue the following command as root:

   ```
   # clientconfig_authority -a <userid>
   ```

   A user ID can be removed from the authorized list by using a **`-d`** flag instead of **`-a`**.
5. Thereafter, the indicated user ID can use the **`clientconfig`** command.

As a practical matter, the same user ID can be selected for both functions. The ability to bypass root use with these commands does not alter the operation of the **`SecureUpdateUtility`**, **`Z1090_token_update`**, **`Z1091_token_update`**, or **`clientconfig`** commands if used by root in the normal way.

Starting with ISV zPDT GA10 Fix Pack 2, the token access functions that are encountered while running ISV zPDT no longer require root access. Some administrative commands, such as the ones in this section, still require root, but routine operation of ISV zPDT no longer requires it. Table 4-1 on page 72 lists the specific commands that require Linux root authority. The OSA emulation function of ISV zPDT no longer requires Linux root access.

[12] This situation applies to zPDT release GA5 and earlier releases.

13.8.2 Gen1 token server monitoring

Although it is not part of the IBM zPDT product, the Gen1 token software that is provided by the company supplying the license tokens and associated software has a web monitoring function. This function is not relevant to normal ISV zPDT operation, but it might be construed as an exposure. You can disable this monitor function as follows:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# cp -p sntlconfigsrvr.xml sntlconfigsrvr.xml.orig          (Make backup.)
#     (edit sntlconfigsrve.xml, find <ConfigureLicenseMonitorPort>
       and change 7002 to 0)
# ./loadserv restart
```

13.9 z1090instcheck

The **z1090instcheck** command should be run after ISV zPDT software is installed and Linux configuration changes are completed. It should be run again after any Linux updates. The output varies somewhat from release to release. Here is an example of the output from **z1090instcheck**:

```
 1. SUSE os level at 11.4 which is greater the minimum level         OK
 2. SUSE kernel.shmmax of 36000000000 is greater than min.        *NOTE*
        shmmax should be greater than 1.1 times the sum
        of z memory (as specified in your devmap) for
        ALL your 1090 instances.
 3. SUSE (kernel.shmall * PAGE_SIZE) is 4722366482869644165120
        which is greater/equal to kernel.shmmax which is          OK
 4. SUSE kernel.msgmni is 512 which is                            OK
 5. SUSE kernel.msgmax is 65536 which is                          OK
 6. SUSE kernel.msgmnb is 65536 which is                          OK
 7. SUSE net.core.rmem_default is 1048576 which is                OK
 8. SUSE net.core.rmem_max is 1048576 which is                    OK
 9. SUSE kernel.core_uses_pid is 1 which is                       OK
10. SUSE kernel.core_pattern is core-%e-%p-%t which is            OK
11. SUSE ulimited -c is set to unlimited                          OK
12. SUSE ulimited -d is set to unlimited                          OK
13. SUSE rpm libstdc++45-32bit-(x86_64) is installed              OK
14. SUSE sntl-sud-7.5.2-0.i386 rpm is greater than required       OK
15. SUSE zpdt-shk-server-1.3.1.2-0.i586 rpm is equal to required level OK
16. SUSE dmidecode-2.11-15.1.x86_64 rpm is greater than required  OK
17. SUSE rpm beagle is not installed which is                     OK
18. SUSE rpm zmd is not installed which is                        OK

Running uimcheck ...

The UIM client is configured in remote mode.

Local Host Name....... w510.itso.ibm.com
Local Serial Number... 32683
Local machine UUID.... 81402112-2551-CB11-A1F3-D9473378A894

Remote server......... 192.168.1.2
Remote server port.... 9451
```

The output details include the following lines:

- Line 1 verifies that you are using SUSE (or Red Hat or Ubuntu) and that it is at an acceptable level.
- Lines 2 and 3 check kernel controls for virtual shared memory. The values that are shown here are examples. The **`shmmax`** value should be at least as large as stated in the note. The **`shmall`** value that is shown is typical of a 64-bit Linux distribution. However, some distributions have this number set much smaller and you might receive a warning message for this line.
- Line 4 (`msgmni`) is appropriate for a reasonable number of ISV zPDT I/O devices.
- Lines 5 and 6 are needed for OSA operation. The exact values are not important but should be larger than the default sizes in most distributions.
- Lines 7 and 8 reflect values for heavy OSA usage or larger frames.
- Lines 9, 10, and 11 reflect parameters for core image files. These files are potentially important if ISV zPDT problems are encountered.
- Line 12 should be set as shown.
- Lines 11 and 12 are for the token modules and verify that the correct levels are present. The levels that are distributed with ISV zPDT should not be replaced with other versions, even if the other versions have later levels.
- Line 13 verifies that 32-bit support is installed with Linux, which is needed by the token modules.
- Line 16 reflects a module that might be useful for debugging. It is not critical.
- Lines 17 and 18 address applications that caused ISV zPDT problems in the past.

Additional checks might be added to later versions of **`z1090instcheck`**. Some checks are absolute, and others look for values in a range that is appropriate. Your output from **`z1090instcheck`** might differ slightly from what is shown here as minor details can change with ISV zPDT updates or new Linux distributions.

The evaluations that are listed by this command might not be *required* for your system. The command output is intended as a starting point only.

13.10 CKD versioning

A function is available to allow an emulated 3390 volume to be “reset” to a selected point in time. This function is known as *CKD versioning*. The usage is as follows:

- A command is issued for selected emulated volumes (which might be all the emulated volumes in your configuration) to enable versioning. This task must be done when ISV zPDT is not active.[13]
- ISV zPDT is started; an IPL of an operating system is done; and the volumes are used normally.
- Later (when ISV zPDT is not running), commands are issued to either commit whatever changes were made to the volumes or restore the volumes to the exact content they had when CKD versioning was enabled.

A typical usage might be for a demonstration or benchmark. After the demonstration or benchmark is completed, the volumes can be restored to their original state.

[13] zPDT can be active if the selected volumes are not in the active devmap.

A CKD-emulated volume can be considered to have two versions. Version zero is the original state of the volume and version one is a volume that was changed after versioning was enabled. Only one restore version is possible for changed volumes, that is, multiple concurrent generations of versions are not possible.

The following commands are associated with CKD versioning:

$ alcckd /z/WORK01 -ve	Enable versioning for the indicated volume.
$ alcckd /z/WORK01 -vr	Restore the volume to the original content.
$ alcckd /z/WORK01 -vc	Commit the changes to the volume.
$ alcckd /z/WORK01 -vi	Inquire about the versioning status of the volume.
$ alcckd /z/WORK01 -vd	Disable versioning (if no changes were made).

These examples use an emulated volume that is stored in `/z/WORK01`. You specify the name of the Linux file containing your volume.

When changes are made to a track of a version-enabled volume, the original track contents are saved at the end of the emulated volume file. Only one “original track” is saved, and subsequent changes to the track update the track within the emulated volume. If the volume is *restored*, the original tracks replace the changed tracks. If the changes are *committed*, the original tracks are discarded. Restoring or committing a volume results in a volume that is not enabled for versioning. It can be enabled again with the **-ve** option.

The version-enabled status of a volume is carried over subsequent ISV zPDT starts and stops, and subsequent operating system IPLs. The version-enabled status remains until the volume is restored or committed. The Linux file size for the emulated volume grows as more “original tracks” are stored. In an extreme case, where every track on the volume is changed, the Linux file grows to twice its original size. In typical cases, relatively few tracks on a volume are changed through normal usage and the Linux file growth is minor.

If versioning was enabled for a volume but there were no changes to the volume, the **-vd** option can be used to disable versioning. If there were changes to the volume (causing the versioning function to start operation), then the **-vd** option is rejected. You must use **-vr** (to restore the volume to the original state) or **-vc** (to commit the changes) to disable the versioning function. The **-vi** (inquiry) option displays the current versioning state of the volume and, if versioning is active, displays the number of CKD tracks that were versioned.

13.11 ISV zPDT messages

Most messages that are issued by ISV zPDT have unique message numbers. The **msgInfo** command can be used to obtain more information about a message. An example of its usage might be as follows:

```
$ awsckmap aprof1                 (Check your devmap.)
AWSCHK200I Checking DEVMAP file 'aprof9' ...
AWSCHK204I Processed 204 records from DEVMAP /home/ibmsys1/aprof1
AWSCHK208I Check complete, 0 errors, 0 warnings detected.

$ 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 is needed, but
AWSINF013I     if errors are present, the DEVMAP cannot be used to start the
system.
```

All message numbers are in the form `AWS`*`cccnnns`*, 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, or Terminal).

The message code that is specified on the **`msgInfo`** command can omit the AWS prefix and the severity code. For example, **`msgInfo chk208`** is sufficient. There is also an environment variable that is named **`Z1090_MSG`** that controls message formatting. It can be set to `FULL` (the default), `CODE` (prints only the message number and no text), `TEXT` (prints the message text and no code), or `SHORT` (drops the AWS prefix on the message number).

13.12 Linux TCP/UDP ports

Several Linux TCP/IP ports are used by normal ISV zPDT operations. In most cases, a default port number is used and you should be aware of these port numbers. When using these ports for connections outside your base Linux machine, you must ensure that any firewall permits the usage of these ports. The port information is as follows:

- The ISV zPDT token device is accessed through TCP/UDP and requires a port number. Linux port 9450 is assigned for this device. The UIM function uses Linux port 9451, which is meaningful only when using a remote license manager.
- The Gen2 license server, if used, uses port 1947.
- If you use the awsctc device manager, it uses Linux port 3088 by default. Additional CTC devices use different port numbers, typically 3089, 3090, and so on.
- The aws3274 device manager (for “local” 3270 connections) typically uses Linux port 3270.
- The migration tool (see Chapter 15, “Direct access storage device volume migration” on page 313) uses port 3990 by default.
- The Server Time Protocol (STP) function uses port number 4567 by default.
- The SafeNet Gen1 license manager provides a status web display page by using port 7002. This manager is not part of the IBM zPDT product, but it is provided with the SafeNet software that manages the token activities.
- If you are operating your base Linux system through VNC, the default ports are 5900+*N*, where N is the VNC display number.

13.13 Remote operation

An ISV zPDT system (including z/OS) can be operated remotely by using a Linux CLI (Telnet, VNC, or SSH) and TN3270e sessions that are connected to the base Linux on the ISV zPDT system. No special techniques or setups are required. As with local operation, connecting the TN3270e session for the z/OS console before doing an IPL of z/OS is important.

Security considerations in your environment determine whether simple Telnet or the more secure SSH should be used for the Linux CLIs that are used to control ISV zPDT.

Complete remote operation of a Linux system running ISV zPDT might be most convenient when using VNC. When using VNC, it is most convenient to start ISV zPDT from a VNC window so that the owner can disconnect the VNC session and reconnect to it later without affecting ISV zPDT operation. Managing the VNC server on the base Linux system typically requires a Linux command-level connection with something like Telnet, SSH, or PuTTY.

13.14 ISV zPDT devices

The current version of ISV zPDT has a maximum of 2048 emulated I/O devices for an instance. If you define more than approximately 100 devices, you might need to change Linux kernel definitions. You might need to alter the following items:

- The total amount of Linux shared virtual memory, which is set by the **`kernel.shmall`** parameter.
- The amount of Linux shared virtual memory that is available to a process, which is set by the **`kernal.shmmax`** parameter.
- The maximum number of shared memory segments, which is set by the **`kernel.shmmni`** parameter.
- The parameter that relates to the number of Linux semaphores that are available, which is the **`kernel.msgmni`** parameter.
- The number of semaphore devices that are available, which is set by the **`kernel.sem`** parameter.

The default values for these parameters appear to vary considerably with various Linux releases. In many cases, the default **`shmall`** and **`shmmax`** parameters are large and do not need to be changed. The other parameters *might* need to be changed.

Each I/O device that is defined in your devmap requires approximately 300 KB of shared virtual memory. The total of this virtual memory, plus the memory that is defined for your IBM zSystems server, must be within the allowed total Linux shared virtual memory. Your defined IBM zSystems server memory size (plus 20% overhead) must fit within the shared virtual memory that is available for a process.

The ISV zPDT developers do not have exact formulas for these parameters. The default numbers that are created by the `/usr/z1090/bin/aws_sysctl` script are reasonable for typical ISV zPDT systems with a few dozen defined devices and around 10 - 20 GB that are defined for IBM zSystems server memory.

We found the following formula for **`msgmni`** to be reasonable:

`kernel.msgmni` = (350 + 3 * number of I/O-devices)

This formula is not exact, but it should produce safe values. Also, if you have more than approximately 128 emulated I/O devices, you should use the **ulimit -m** and **-v** statements that are described in Chapter 5, “ISV IBM Z Program Development Tool installation” on page 125.

Using multiple ISV zPDT instances, each with many devices, might exhaust the semaphore space in Linux, which results in a hang when starting an ISV zPDT instance. This issue might be addressed by another kernel parameter:

```
kernel.sem 250 32000 32 128        (default values in some distributions))
kernel.sem 250 32000 250 1024      (You might try these values.)
```

Unusually large numbers of I/O devices (across multiple ISV zPDT instances), generally in excess of a total of more than about 1200 devices, can require a higher **shmmni** value:

```
kernel.shmmni 8192                  (Default is typically 4096.)
```

In one rather large case, by using 2048 devices in a single ISV zPDT instance, we defined the following values:

```
kernel.shmmall=4300000000
kernel.shmmax=10000000000 (Make larger for larger IBM zSystems server memory.)
kernel.shmmni=32000
kernel.msgmni=6500
```

These numbers might not be optimal, but they worked for us.

13.15 Startup scripts

We created several startup scripts for our z/OS ISV zPDT system, such as this example:

```
$ gedit start00
   cd /home/ibmsys1
   awsstart aprof1
   sleep 6
   echo ISV zPDT started
   x3270 -port 3270 mstcon@localhost &
   sleep 2
   x3270 -port 3270 tso@localhost &
   sleep 5
   ipl a80 parm 0a8200
   sleep 2
   echo IPL issued
```

Using this script, you can start your system with a single **./start00** command. The scripts (differing only in the IPL parm data) are trivial and can be enhanced in many ways. Be certain to allow sufficient time for the 3270 sessions to start before running the IPL command.

A better alternative to a Linux startup script is to embed Linux commands in the ISV zPDT devmap, as described in 3.2, “System stanza” on page 50.

13.16 Suspension and hibernation

Laptop PCs with Linux software offer suspend ("save to RAM") and hibernation ("save to disk") functions to save the current PC state and move to a non-operational state that uses little power. ISV zPDT is *not* tested for and does *not* support these functions. These functions become problematic when timed token drivers or time-dependent LAN connections are involved, such as with ISV zPDT license servers.

Some ISV zPDT users reported using these functions successfully. One technique is to use a z/OS **`quiesce`** command before the suspend operation, which allows z/OS to complete in-progress I/O and prepare other internal operations for a restartable disabled wait. After restarting the PC, an ISV zPDT **`restart`** command (entered in the same ISV zPDT operations window that was visible before the hibernation) should cause z/OS to resume operation.

We had erratic experiences with this approach on different PCs and different Linux versions. In one case, using Fn-7 after "opening the lid" restored the graphic window and the **`restart`** command worked as expected. In another case, we could not restart the graphic window, but in yet another case the graphic desktop restarted by itself.

Some users reported that the token driver or license manager did not restart. A common symptom is an "invalid license" or "heartbeat missing" for one or more CPs.[14] Linux commands such as the following ones might help:

- **`# service shk_usb restar`**t (older Linux)
- **`# service sentinel_shk_server restart`** (older Linux)
- **`# systemctl start sentinel-shk-usb.service`** (more recent Linux systems)
- **`# systemctl start sentinel-shk-server.service`** (more recent Linux systems)

The **`systemctl`** command might be more appropriate than **`service`**, depending on the Linux distribution. The action parameter can be **`start`** or **`restart`**.

You must determine whether hibernation or suspension works for you. It is a Linux function, and *not* part of ISV zPDT.

[14] After these error messages, zPDT might acquire the missing license automatically. If this event happens, z/OS is probably unaffected and continues to run. The error messages are seen only in the Linux CLI that is used to start zPDT.

13.17 Channel measurement blocks

With ISV zPDT GA9 and later, certain fields in channel measurement blocks are meaningful for ISV zPDT users, as shown in Figure 13-1. As specified in the IBM zSystems architecture, the unit of timing is 128 microseconds.

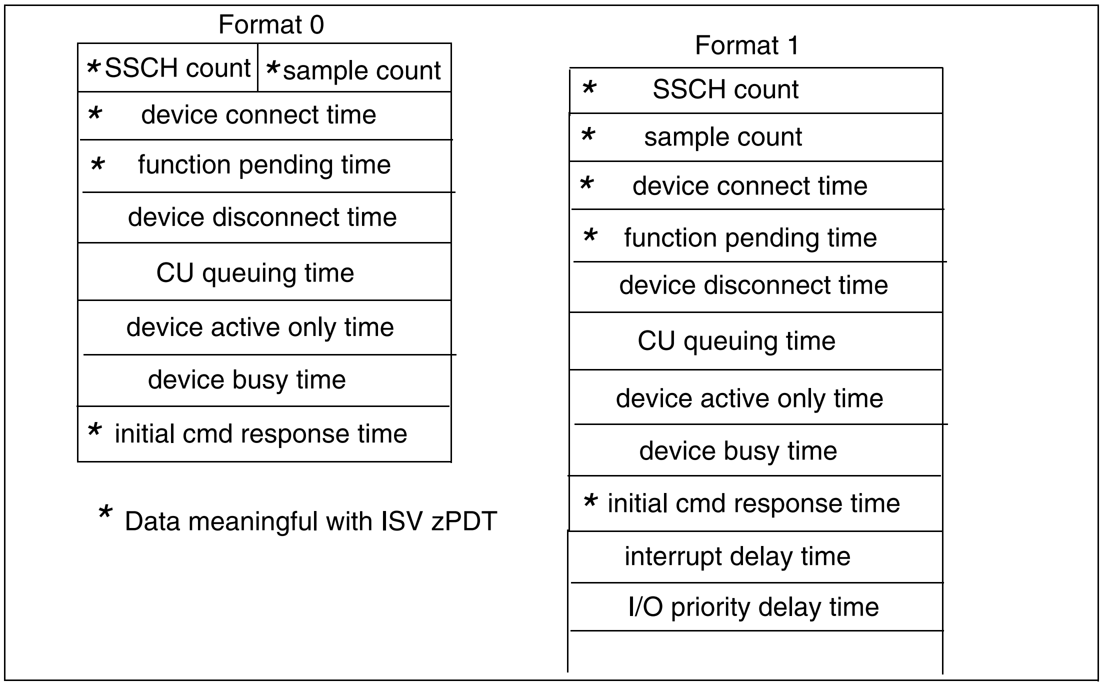


Figure 13-1 Channel Command Blocks

Equivalent fields in a subchannel extended measurement word have a granularity of 0.5 microseconds. The data in the measurement blocks (and measurement word) might not match what is found on a large IBM zSystems server. Various Linux cache and ISV zPDT buffer operations (not visible to z/OS) can result in measurement times that appear impossibly short for a standard IBM zSystems server.

13.18 x3270 scripting

The x3270 emulator has a scripting function that is not well documented. Web searches for “x3270 scripting” and “x3270if” are good starting points. The following partial script starts an x3270 session (so you can follow the action by watching it) and logs on to TSO. This script[15] is entered into a simple Linux file in your home directory, and this file is later run.[16]

```
#!/bin/bash
x3270 -socket &                         (Start the x3270 session.)
pid=$!                                  (no internal spaces here)
until x3270if -p $pid ''''> /dev/null      (four single quotation marks without
spaces)
do sleep 0.1
done
x3270if -p $pid 'connect(“localhost:3270”)'
x3270if -p $pid 'wait()'          (wait until 3270 unlocks at writable field)
x3270if -p $pid 'string(“logon ibmuser”)'
sleep 3                                 (optional; helps watching action)
x3270if -p $pid 'enter'
x3270if -p $pid 'wait()'
sleep 3                                 (optional)
x3270if -p $pid 'string(“xxxxxxxx”)'       (the TSO password)
x3270if -p $pid 'enter'
x3270if -p $pid 'CloseScript()'         (See following note.)
```

The **CloseScript** functions leave the x3270 session running. You can manually enter whatever TSO commands that you want to use. The **wait** function is blocked until the 3270 keyboard is unlocked and the cursor is positioned on a writable field. The **quit** function (not documented) ends the x3270 session.

The scripting function provides multiple ways to save the 3270 panel output, but it does not provide a way to wait for a particular output string. A user can save the current 3270 panel (in ASCII character format) and use Linux shell commands to search the saved data for whatever keywords are needed, but this method quickly turns into a substantial effort in Linux shell programming.

Among the other functions that are documented for x3270 scripting is the **MoveCursor(row,col)** function. The **row** and **col** parameters in this function are zero-based, and the row and column numbers that are displayed at the bottom of an x3270 panel are one-based.

13.19 Scenarios for awstape

The practical usage of the ISV zPDT awstape device manager can be a bit confusing to users without hands-on experience with mainframe tape units. The physical mounting of a tape volume on a tape drive, and the reading, writing, and positioning of the “tape” that is involved, must be emulated. The ISV zPDT **awsmount** command emulates the physical mounting of a tape volume and the ISV zPDT **ready** command emulates a tape drive (with a mounted tape) going from `not-ready` to the `ready` state. Both of these commands produce an emulated interrupt that informs the IBM zSystems operating system of the action.

[15] We thank Paul Mattes, the primary author of x3270, for help with bypassing a timing problem while producing this script.
[16] Use a **chmod** command to make the Linux file an executable file.

There is only one awstape format. The IBM zSystems operating system (working with devmap definitions) handles an emulated tape drive as an IBM 3420, 3480, 3490, or 3590 drive, with appropriate commands for the specified drive type. However, the Linux file that is created (containing the emulated tape volume) is the same in all cases. For example, you can create an awstape volume by using an emulated 3420 drive and later read it by using an emulated 3490 drive.

The following scenarios describe several approaches to using awstape.

13.19.1 Creating a labeled tape volume

Consider the following job that calls for a standard label tape volume with the volume serial TAPE99. Furthermore, assume that we do not have an awstape volume with volser TAPE99.

```
//OGDEN30 JOB 1,OGDEN,MSGCLASS=X
// EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD UNIT=561,VOL=SER=TAPE99,DSN=MY.TAPE,DISP=(NEW,KEEP)
//SYSUT1 DD DISP=SHR,DSN=OGDEN.CARD.SEQ
```

The z/OS console interaction for this job might be as follows:

```
JOB01855  IEF403I OGDEN30 - STARTED - TIME=14.32.27
JOB01855 *IEF233A M 0561,TAPE99,,OGDEN30,,MY.TAPE
      (In a Linux CLI, enter awsmount 561 -m TAPE99.aws.)
JOB01855  IEC512I LBL ERR 0561,      ,NL,TAPE99,SL,OGDEN30,,MY.TAPE
JOB01855 *IEC704A L 0561,TAPE99,SL,NOCOMP,OGDEN30,,MY.TAPE
JOB01855 *09 IEC704A REPLY 'VOLSER,OWNER INFORMATION','M'OR'U'
JOB01855  R 9,TAPE99
JOB01855  IEC705I TAPE ON 0561,TAPE99,SL,NOCOMP,OGDEN30,,MY.TAPE,MEDIA1
JOB01855  IEC205I SYSUT2,OGDEN30,FILESEQ=1, COMPLETE VOLUME LIST,  066
                 DSN=MY.TAPE,VOLS=TAPE99,TOTALBLOCKS=1
JOB01855  IEF234E K 0561,TAPE99,PVT,OGDEN30
```

The IEF233A message requests that the operator mounts tape volume TAPE99 on tape drive 0561. Then, the job waits until the operator mounts the requested tape. The ISV zPDT user must enter an **awsmount** command in a Linux CLI. This action satisfies the z/OS mount request. In the example here, we specified Linux file name TAPE99.aws.[17] This Linux file name is arbitrary, and it does not need to contain the tape volser or an aws suffix. As a best practice, use a naming convention. In this example, file TAPE99.aws is created in the Linux directory that was used to start ISV zPDT. Alternatively, a complete Linux path name can be specified.

In this example, after the **awsmount** command is issued z/OS attempts to read the standard label but there is nothing in the file. This situation outputs the z/OS messages that are shown and allows the z/OS operator to enter the volume serial number. After this task is done, the job runs and presumably writes data on the tape. At the end of the job, the tape is [logically] dismounted, as shown by the IEF234E keep message.

The **-m** flag in **awsmount** indicates a simple mount function. If there is already a tape volume that is logically mounted on the drive, an **-o** flag specifies that whatever emulated tape file is mounted is replaced by the newly specified file. There are many more **awsmount** options that are described in 4.2.18, “The awsmount command” on page 84.

[17] In this example, the Linux file does not yet exist. It is created by the **awsmount** command.

13.19.2 Quick SL volume creation

You can use the **aws_tapeInit** command to create a standard label awstape volume with no data and with minimal label information. Here is an example:

```
$ aws_tapeInit tape01 /tmp/mywork/TAPE01
```

This command creates a Linux file (/tmp/mywork/TAPE01) in awstape format with the standard label TAPE01. (The conversion of the volume serial to uppercase Extended Binary Coded Decimal Interchange Code (EBCDIC) is automatic.) The new volume is a Linux file, and the command does not perform a mount function.

13.19.3 Using an existing labeled tape

If you run the **IEBGENER** job in the previous example and want to use an existing awstape volume, the console interaction might be as follows:

```
JOB01855 *IEF233A M 0561,TAPE99,,OGDEN30,,MY.TAPE
      (In a Linux CLI, enter awsmount 561 -m TAPE99.aws.)
JOB01855  IEC205I SYSUT2,OGDEN30,FILESEQ=1, COMPLETE VOLUME LIST,  066
                DSN=MY.TAPE,VOLS=TAPE99,TOTALBLOCKS=1
JOB01855  IEF234E K 0561,TAPE99,PVT,OGDEN30
```

In this case, the TAPE99.aws in the **awsmount** command should specify the Linux file that holds the awstape volume.

13.19.4 Automatically premounting an existing labeled tape

We can use the ISV zPDT awscmd device manager (described in Chapter 11, “The awscmd device manager” on page 227) to issue the **awsmount** command for an existing awstape volume. An example is as follows:

```
//MNTCMD JOB 1,OGDEN,MSGCLASS=X
// COMMAND 'VARY 561,ONLINE'
// COMMAND 'UNLOAD 561'
//* Use either PARM or SYSIN to specify the awsmount command.
//AWSCMD EXEC PGM=AWSCMD
//STEPLIB DD DISP=SHR,DSN=OGDEN.LIB.LOAD
//SYSPRINT DD SYSOUT=*
//TAPE DD UNIT=(560,,DEFER),LABEL=(1,BLP),VOL=SER=123456,DSN=X
//SYSIN DD *
awsmount 561 -m /z/zos23/TAPE99
//*
//  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.ASM
//SYSUT2 DD DISP=NEW,VOL=SER=TAPE99,UNIT=561,DSN=OGDEN.TAPE.DATA
```

While working with this technique, we noted the following points:

- The **VARY** and **UNLOAD** commands (in lines 2 and 3 of the example) were necessary to avoid device status conflicts.
- The AWSCMD step mounts a standard label tape volume (by using the **awsmount** command) that z/OS has not yet requested. This action invokes the z/OS automatic volume recognition (AVR) function, making the volume available for a subsequent request for the volume.
- The example uses in-stream SYSIN to provide the **awsmount** command. A more general approach would point to a data set containing the **awsmount** command, which might have been generated by an earlier job or job step.
- The **TAPE DD** statement in the AWSCMD job step points to the “dummy” tape that is used by the **awscmd** function. For more information, see Chapter 11, “The awscmd device manager” on page 227.

This technique requires experimenting with the setup and JCL to make it work. However, this general approach might be considered by ISV zPDT users working toward an automatic tape library function.

13.19.5 Using special channel command words to manipulate tape volume

zPDT GA9 and later provides two special channel commands for working with awstape. These functions were requested by many ISV zPDT customers. The active devmap must specify the **--MNTCCWS** option on the awstape **name** statement to enable this function.

Channel command 4B is used to change the Linux file that is associated with an emulated tape unit. Channel command 54 is used to obtain the Linux file name that is associated with an emulated tape unit. These operations are not reflected on the IBM zSystems operating system in any way: They do *not* create device interrupts, and they *cannot* satisfy a z/OS operating system mount request.

These channel command words (CCWs) are of limited use with z/OS because z/OS wants to control all device mounts and dismounts, and attempts to read standard labels even if JCL specifies an unlabeled tape (UNLABB). Other IBM zSystems operating systems might have more permissive tape drive usage and might find these CCWs more useful.

A slightly complicated example of using these CCWs with z/OS is as follows:

```
//TAPECCW2 JOB 1,OGDEN,MSGCLASS=X
//  EXEC  ASMACLG,PARM.C='NOXREF',PARM.L='NOLIST,NOMAP'
//C.SYSIN  DD *
TAPECCW  CSECT
         STM   14,12,12(13)    Save caller's registers
         LR    12,15           Use entry-point as base register
         USING TAPECCW,12
         LR    2,13            Get @caller's savearea
         LA    13,SAVEAREA     Get @my savearea
         ST    2,SAVEAREA+4    Chain old to new
         ST    13,8(2)         Chain new to old
* Open tape drive.
A        OPEN  STAPEO          Assume BLP on DD statement
         TM    STAPEO+48,X'10' Did OPEN work?
         BZ    ERROR1          If not, branch
* Write the name of th target awstape file (the name is coded as a constant in
this example).
```

```
D        MVC   SECB,=F'0'      Zero your ECB
         LA    1,SCCMOUNT      @(CCW)
         ST    1,SIOB+16       Place address in IOB
         LA    2,L'PARM        Get length of Linux file name
         STH   2,SCCMOUNT+6    Put length in CCW
D1       EXCP  SIOB            Run the CCW
         WAIT  ECB=SECB        Wait for it
D2       TM    SECB,X'20'      Completed OK?
         BZ    ERROR2          If not, branch
* Read the Linux file name that is mounted (to verify the mount).
C        MVC   SECB,=F'0'      Zero your ECB
         LA    1,SCCRNAME      @(CCW)
         ST    1,SIOB+16       Place address in IOB
C1       EXCP  SIOB            Run the CCW
         WAIT  ECB=SECB        Wait for it
C2       TM    SECB,X'20'      Completed OK?
         BZ    ERROR3          If not, branch
CLOSE    CLOSE (STAPEO)
RETURN   L     13,4(13)        Get @(caller's save area)
         LM    14,12,12(13)    Restore caller's registers
         SR    15,15           Assume return code 0
         BR    14              Exit
*   Problems
ERROR1   ABEND 1               OPEN failed for TAPE
ERROR2   ABEND 2               Error with mount CCW
ERROR3   ABEND 3               Error with query CCW
*
*  Constants and work areas
*
SAVEAREA DC    24F'0'
STAPEO   DCB   DSORG=PS,MACRF=(E),DDNAME=TAPE,DEVD=TA
PARM     DC    C'/z/zos23/UNLABC'  a Linux awstape file with this name
BUFF1    DC    CL64' '         Buffer for reading current file name
         DS    D'0'
SCCMOUNT DC    X'4B',AL3(PARM),X'20',AL3(0)       Mount awstape
SCCRNAME DC    X'54',AL3(BUFF1),X'20',AL3(64)     Read Linux name
SECB     DC    F'0'
SIOB     DC    X'43000000'      IOBFLAG1, FLAG2,SENSE
         DC    A(SECB)          A(ECB)
         DC    2F'0'            CSW (& Residual count)
         DC    A(0)             A(CCW)
         DC    A(STAPEO)        A(DCB)
         DC    2F'0'
         LTORG
         END
/*
//G.TAPE  DD UNIT=(561,,DEFER),LABEL=(1,BLP),DSN=X,
//           VOL=(,RETAIN,,,SER=UNLABB)
//*
//  EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD SYSOUT=*,DCB=(LRECL=50,BLKSIZE=50,RECFM=F)
//SYSUT1 DD UNIT=561,LABEL=(1,BLP),DSN=X,VOL=(,RETAIN,,,SER=UNLABB),
//   DCB=(LRECL=50,BLKSIZE=50,RECFM=F)
```

The flow of this job might not be obvious. Here are some details about it:

- z/OS requires that a device (an emulated tape drive in this case) must be "open" before it can be accessed.[18]
- The **TAPE DD** statement in the first job step specifies a tape volume (UNLABB). When the job is run, the ISV zPDT user must use the ISV zPDT **awsmount** command to mount an unlabeled awstape volume to satisfy the z/OS mount request. The tape volume that is mounted this way is not used. Instead, it is needed to allow the **OPEN** function to complete so that commands can be sent to the tape drive.
- The TAPECCW program (in the example) uses the CCWs to change the Linux file name that contains the awstape volume. In this example, the new Linux file name is coded in the program (`/z/zos23/UNLABC`), but a more realistic approach probably would have the name in another data set.
- The mount CCW (4B) is followed by the query CCW (54). The query reads the same name that was mounted. However, the query must be coded and cannot be command-chained to the mount CCW. Failure to follow the mount with the query results in an `ABEND 214-08` when a **CLOSE** is issued.[19]
- Conversion of the Linux file name from EBCDIC to ASCII (for mount) and ASCII to EBCDIC (for query) is automatic.
- The second job step, an **IEBGENER** utility that represents an application program, has a **DD** statement specifying the same UNLABB that was specified in the first step and passed to the second step through the **RETAIN** option. However, the first step changed the actual file to be used (`/z/zos23/UNLABC`).
- This mount CCW (4B) does not perform a z/OS mount, that is, there is no mount interrupt that is generated and z/OS is unaware that the awstape volume name was changed in the background.
- The new Linux awstape file that is mounted is positioned at the beginning of the file but is not opened (at the Linux level of `open`) until a read or write command is directed to it.
- The mount CCW (4B) points to the Linux file name. The length field in the CCW must be correct and not include trailing blanks with the Linux file name. The length field for the query command in the example is arbitrary. You probably have a buffer longer than needed and use the residual count from the I/O buffer to determine the actual length of the returned data. In the example, the query data is not examined.
- The general flow that is described here is unlikely to work for labeled tapes in z/OS.

These CCWs were requested by several ISV zPDT users. Some experimentation might be needed to effectively use them.

13.19.6 Premounted tape

It might be convenient to mount an emulated tape that is ready to use without MVS console interactions after z/OS undergoes an IPL. There are several ways to approach this task depending on your exact requirements. As a starting point, assume you have a standard labeled tape volume (an "awstape" volume) in `/home/ibmsys1/TAPE01`. Assume that the tape (with a tape label) was previously created. In these examples, we assume that your z/OS IODF defines address 561 as a 3480 device.[20] There is nothing unique to 3480 drives in these examples: We use them to present concrete definitions and commands.

[18] There are techniques for working around this requirement, but these techniques require authorized programs working with z/OS interfaces that are not intended for external use.

[19] This situation was true at the time of writing, and the zPDT developers are not clear about why z/OS creates this `ABEND`.

[20] This configuration is consistent with z/OS ADCD releases.

Complete the following steps:

1. You can include the tape in your devmap definitions. For example:

   ```
   [manager]
   name awstape 3000
   device 561 3480 3480 /home/ibmsys1/TAPE01
   ```

   After starting ISV zPDT and performing an IPL of z/OS, you can run one job by using this volume without encountering any z/OS mount messages. The job should contain a **DD** statement like the following one:

   ```
   //SYSUT2 DD DISP=(OLD,KEEP),VOL=SER=TAPE01,UNIT=561,DSN=MY.DATA
   ```

   The **DISP** also could be `NEW`. After the first z/OS job that uses this tape completes, the tape is logically dismounted and more uses produce z/OS console **MOUNT** messages. In general, you respond to these messages through a Linux CLI with the following command:

   ```
   $ awsmount 561 -m /home/ibmsys1/TAPE01
   ```

2. Instead of including the tape file name in the device statement, you can include a command in the devmap, such as:

   ```
   [system]
   memory 8G
   processors 3
   3270port 3270
   command 2 x3270 localhost:3270
   command 2 awsmount 561 -m /home/ibmsys1/TAPE01
   ...
   ...
   [manager]
   name awstape 3000
   device 561 3480 3480
   ```

   The usage is the same as in the previous example, that is, the tape is automatically mounted for the first (and only the first) job that calls for it.

3. After z/OS undergoes an IPL[21] but before the z/OS job is submitted, you can enter the following command through a Linux CLI:

   ```
   $ awsmount 561 -m /home/ibmsys1/TAPE01
   ```

   This command provides an unsolicited mount and triggers the AVR function in z/OS. The next z/OS job that calls for the tape volume immediately runs without producing any MVS console **MOUNT** messages.

4. You can issue a **mount** command from the MVS console:

   ```
   MOUNT 561,VOL=(SL,TAPE01)
   ```

   This command produces a mount message on the MVS console. Enter the **awsmount** command through a Linux panel:

   ```
   $ awsmount 561 -m /home/ibmsys1/TAPE01
   ```

   This usage differs from the other examples in that the tape volume remains mounted after the jobs that are using it end. You can repeatedly submit jobs that use the volume without producing mount messages on the MVS console. (This same technique can be used for a non-labeled tape.)

[21] Assuming that you have tape drive 561 in you devmap, as in the previous example.

5. Many z/OS systems invoke the VTAMAPPL program during system startup. This program runs a series of MVS commands (a “script”). The **MOUNT** command can be included in this script. For example, a recent z/OS ADCD system has a script in parmlib member VTAM00 that can be modified like the following stanza:

```
S RRS,SUB=MSTR
S TSO
...
...
PAUSE 2
VARY 561,ONLINE
MOUNT 561,VOL=(SL,TAPE01)
PAUSE 2
VARY 561,ONLINE
```

   This example assumes that you included the file name (`/home/ibmsys1/TAPE01` in our examples) in the **device** statement in the devmap. This method results in the tape staying mounted for use in multiple jobs. The second **vary 561,online** command might be necessary to complete the mount. In some cases, the mount remains pending after **vary 561,online**. In this case, a **ready 561** command from a Linux console completes the mount.

13.19.7 The awstape utilities

ISV zPDT users sometimes build a substantial “tape” library on disk, which is all in awstape format. The **tapeCheck** command can be used to verify that a file (that corresponds to a tape volume) is in the correct awstape format:

```
$ tapeCheck /z/TAPE01                  (Verify th format of the awstape file.)
```

The **tape2file** command copies an awstape file to a simple byte stream in a Linux file and removes the awstape control blocks within the file:

```
$ tape2file /z/mytape /tmp/filex
```

The **card2tape** command copies a Linux text file (in ASCII or EBCDIC) to an awstape file as 80-byte records by using the same conversion conventions as the awsrdr device manager:

```
$ card2tape /tmp/myLinux.stuff /z/tape01      (Copy without conversion.)
$ card2tape -a /tmp/myLinux.xyz /z/tape01     (Force conversion of ASCII to
EBCDIC.)
```

The **tape2tape** command copies an emulated tape volume (an awstape file in Linux) to another emulated tape volume (another awstape file in Linux):

```
$ tape2tape /tmp/old.tape /z/new.tape
$ tape2tape -i -s /tmp/old.tape               (Scan and summarize tape content.)
$ tape2tape -c /tmp/old.tape /mine/new.tape   (Copy and compress.)
```

The **-s** flag (scan flag) prevents the creation of an output file. The **-i** flag displays a summary of the contents of the input tape. This command is normally used to compress or decompress an awstape volume or to scan the content. A simple copy of an emulated tape volume (without any additional processing) is easily done with the Linux **cp** command.

The **tapePrint** command lists the contents of an emulated tape volume. Data is displayed in hex and character format. The characters are assumed to be EBCDIC unless the **-a** flag is used:

```
$ tapePrint /tmp/my.awstape.file
$ tapePrint -a /tmp/my.awstape.file | more
```

Both the **card2tape** and **tape2tape** commands can produce compressed awstape files.

14

Tape drives and tapes

Starting with Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) GA10, general SCSI tape drives are no longer “officially” supported. Only Fibre Channel-connected 3590 drives are supported. However, the awsscsi device manager from earlier ISV zPDT releases continues to be distributed *as is*. You can continue to use it, but IBM cannot address problems with general SCSI tape drives. The awstape device manager remains a standard part of ISV zPDT.

14.1 ISV zPDT 359x tape support

Information Technology Company (ITC)[1] works with ISV zPDT development by testing changes to the awsscsi device manager that are designed to more fully support Fibre Channel Protocol (FCP)-attached IBM 3590/3592 tape units. Although these devices can be physically attached to an ISV zPDT system before these awsscsi enhancements, they function only in basic mode by emulating a 3490 device. With updated support, these units can operate in full 359x mode, with minor exceptions.[2]

The FCP adapters

ITC tested several FCP adapters, from both Emulex and QLogic, including older 2 Mb adapters such as the Emulex LPe1150-E, up to the current FCP adapters that are offered with Lenovo System x, such as the 8 Mb Emulex, IBM part number 42D0485. Although most testing was with Emulex adapters because they were more available, we did test with at least one QLogic adapter (dual-core 8 Mb, IBM part number 42D0510). In all cases, the FCP driver that was included in openSUSE 12.2 functioned correctly and newer or different drivers were unnecessary.[3] Firmware-related issues, even on the oldest adapters, did not occur, so the firmware levels of the various adapters are not documented here. All adapters that were tested were PCIe format adapters.

14.1.1 3590/3592 tape drives

Testing was performed with 3592-J1A and 3592-E05 tape units, and the 3590-H11 with a 10-cartridge autoloader.[4] There are no configuration options for 3590 devices. These drives are also known as TS1120 models.

All 3592 drives must be configured[5] for RACK installation, and not for LIBRARY installation. Each drive has two ports. The port that you intend to use must be configured as `PORT SPEED=Auto Negotiate`, `TOPOLOGY=L-Port`. As a best practice, set `HARD ADDRESS` to `x88`, although it is not required.

Performance is limited by the x86 architecture of the underlying hardware platform and the configuration of the Linux operating system. Therefore, performance numbers cannot be expressed in meaningful terms. Our experience showed that the performance is acceptable for development users, and allows fully compatible interchange of media with other mainframe data centers.

Important: Although you might have a 3592 physical SCSI drive, it should be defined as a 3590 in the device map (devmap) to match a z/OS input/output definition file (IODF) entry, as in this example:

```
device 590 3590 3590 /dev/sg3
```

[1] Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042. For more information, contact sales@itconline.com.

[2] One exception is that automated tape library functions are not supported.

[3] This situation might not be true for other or older Linux distributions or for FCP adapters that are not widely used.

[4] ITC also has single and dual drive models (E05, E06, and E07) that are available and complete ready-to-run units with power supplies, an IBM FICON adapter, a FICON cable, a CE panel, and a warranty.

[5] Configuration changes are entered through an optional CE panel. The drives that are obtained by the ISV zPDT developers did not require configuration changes. You might discuss the need for a CE panel with your 359x provider.

14.2 Discontinued support

Important: The remainder of this chapter describes SCSI usage that is untested and without formal support, starting with ISV zPDT GA10. This text is included for customers who might still be using SCSI tape drives.

SCSI tape drives might be used in two ways:

- With the awsscsi device manager, SCSI tape drives may be used by IBM zSystems programs.
- Several Linux utilities that directly use SCSI tape drives are provided with ISV zPDT. These utilities can be used when ISV zPDT is not active. These utilities are not associated with devmaps or a device manager.

However, not all SCSI tape drives are usable. Their usability depends on the exact tape drive model, the firmware level, the firmware options that are selected, the exact SCSI adapter (and firmware level), and the exact cable characteristics. Several different tape drives were tested in previous years, but we cannot guarantee that *your* tape drive will work. As a best practice, work with your ISV zPDT supplier to understand your SCSI tape drive environment.

This chapter describes three categories of SCSI tape drives:

- Tape drives that use traditional SCSI cables. These drives are emulated as 3420, 3480, or 3490 drives, regardless of the actual nature of the tape drive.
- IBM 359x tape drives that are connected with fiber cables that are defined as 3420, 3480, or 3490 drives in the devmap. These drives are subject to the same comments as when using parallel SCSI cables.
- IBM 359x tape drives that are connected with fiber cables that are defined as 3590 drives in the devmap.

The awsscsi device manager

Selected SCSI tape drives can be used as “real” tape drives during ISV zPDT operation. The awsscsi device manager is used so that the SCSI tape drive can appear as a 3420, 3480, 3490, or 3590[6] devices.

A typical devmap definition might be as follows:

```
[manager]
name awsscsi 7000
device 581 3490 3490 /dev/sg5
```

The `7000` in this example is the **`CUNUMBR`** operand. This example defines a tape drive at address (device number) 581. If z/OS is used, then the z/OS IODF must have a corresponding device (IBM 3490) that is defined for this address. (z/VM detects devices dynamically and finds a 3490 at this address.)

The *appearance* to software (as a 3490 in the example above) has no direct relationship to the SCSI device type. In this case, `/dev/sg5` might be a digital linear tape (DLT) drive, for example, that has no physical characteristics of a 3490 drive.[7]

[6] Usage as a 3590 drive requires a fibre adapter and a “real” 359x drive.
[7] IBM did not formally test DLT drives.

As shown in Figure 14-1, Linux SCSI devices can be addressed through two interfaces. The relationship between the `st` and the `sg` interfaces can be complicated because the sequence number that is used for a drive is typically not the same number.

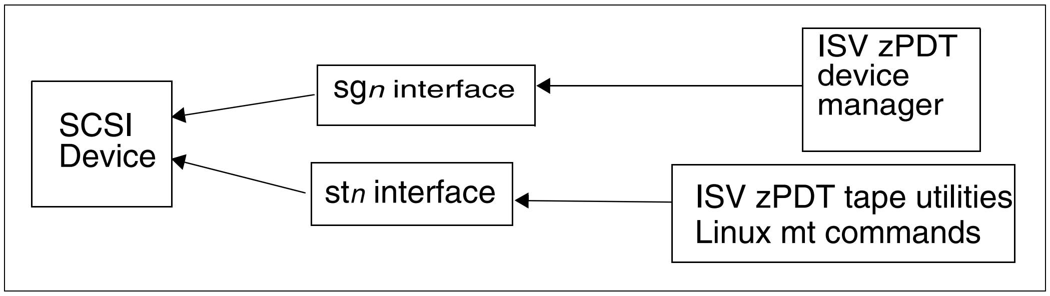


Figure 14-1   SCSI driver interfaces

The Linux device for the SCSI tape drive can be changed with the **`awsmount`** command, as shown in this example:

```
$ awsmount 580 -u /dev/sg5          (Disassociate sg5.)
$ awsmount 580 -m /dev/sg3          (Mount a different drive.)
```

The last operand of the device statement (`/dev/sg5` in the example) denotes the SCSI device that is used. Determining this operand is a bit complicated. Linux can address a SCSI tape drive in three ways:

- `/dev/stN` (Where N starts at 0 and is incremented as needed.)
- `/dev/nstN`
- `/dev/sgN`

The `/dev/stN` and `/dev/nstN` interfaces are for *sequential tape devices*. The first tape drive on a Linux system is `/dev/st0`; a second tape drive would be `/dev/st1`; and so forth. The two forms, `/dev/stN` and `/dev/nstN`, differ only in whether a rewind is performed when the device is closed.[8] The `/dev/stN` and `/dev/nstN` interfaces are used with the ISV zPDT stand-alone tape utility functions, such as **`scsi2tape`** and **`tape2scsi`**, and also by Linux utilities such as **`tar`** and **`mt`**. The `/dev/stN` and `/dev/nstN` interfaces are *not* used with the awsscsi device manager.

The `/dev/sgN` devices are *general SCSI devices*, and the awsscsi device manager uses the `/dev/sgN` interfaces.[9] Unfortunately, the N value for a device is typically not the same in the `stN` and the `sgN` forms. All Linux SCSI devices are assigned `sgN` numbers, and Linux treats many of its normal devices as SCSI (even if they are not really hardware SCSI devices).[10] The first (and only) tape drive on a Linux system would be `/dev/st0`, but it might be `/dev/sg7`, for example.

8 The `/dev/stN` device automatically rewinds when the device is closed. The `/dev/nstN` devices do not provide an automatic rewind.

9 The reason is that the `/dev/stN` interface does not accept the full SCSI command set that is necessary for some tape operations. The `/dev/sgN` interface allows all SCSI tape commands to be passed to the tape drive.

10 The `sg` numbers are assigned in ascending order by the bus and by the device on the bus. However, because Linux treats a number of non-SCSI devices as though they were SCSI, these bus and device numbers are difficult to predict in advance.

Determining the st and sg numbers

To manually determine these interfaces, list /proc/scsi/scsi, as in this example:

```
$ cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00     (this is sg0)
  Vendor: IBM      Model: root            Rev: V1.0
  Type:   Direct-Access                   ANSI SCSI revision: 02
Host: scsi2 Channel: 00 Id: 00 Lun: 00     (this is sg1)
  Vendor: IBM      Model: 03592E05        Rev: 1C91
  Type:   Sequential-Access               ANSI SCSI revision: 03
```

This example shows two SCSI devices (a disk and a tape) that correspond to /dev/sg0 and /dev/sg1. In this case, /dev/sg1 is the device that you specify for the awsscsi device manager, but any Linux utility uses /dev/st0 or /dev/nst0 for the same tape drive. The devices that are listed in /proc/scsi/scsi are in sgN order: sg0, sg1, sg2, and so forth.

You can list both the st and sg interface numbers for all SCSI tape drives that are connected to your system with the **aws_findlinuxtape** command:

```
$ aws_findlinuxtape
Vendor: FUJITSU Model: M2488E  M2488  Rev: 7 C <===>  /dev/st0  /dev/sg10
Vendor: IBM     Model: 03592E05       Rev 1C91  <===> /dev/st1  /dev/sg11
```

The sg number can change from one Linux start to another one if there are any configuration changes.

Permissions

There is another issue with both /dev/stN and /dev/sgN devices: The permission bits must able to be used by ISV zPDT. Some Linux systems allow general user access to /dev/stN devices by default, but no Linux systems allow general access to /dev/sgN devices by default. You must change the permissions to allow general access to your device, as in this example:

```
$ ls -al /dev/sg*
crw-r----- 1 root disk 21, 0 2008-09-03 14:44 /dev/sg0
crw-rw---- 1 root tape 23, 0,2008-99-03 14:44 /dev/sg1
$ su
(Enter root password.)
# chmod 666 /dev/sg1
# exit
$ ls -al /dev/sg*
crw-r----- 1 root disk 21, 0 2008-09-03 14:44 /dev/sg0
crw-rw-rw-+1 root tape 23, 0,2008-99-03 14:44 /dev/sg1
```

In this example, we changed the permissions for /dev/sg1 so that everyone (including ISV zPDT) can read/write to it. This change (by using **chmod**) is lost when Linux restarts, but it is suitable for many situations. Always verify the correct sgN number first by listing /proc/scsi/scsi.

A persistent change can be made by modifying Linux start functions. Unfortunately, there are two problems with this approach:

- The method of modifying Linux start functions differs with different Linux distributions. For example, you might change `/etc/rc.local`, `/etc/rc.d/rc.local`, `/etc/init.d/boot.local`, or another file depending on the exact Linux distribution.
- You can easily make a general change for `stN` and `nstN` devices, but you cannot easily make a general change for `sgN` devices. You should *not* change the permissions to allow universal access to all `/dev/sgN` devices because of security reasons.

You could place the following lines in `/etc/init.d/boot.local` (assuming that your particular Linux distribution uses this file):

```
chmod 666 /dev/st[0-9]            Change all tape devices.
chmod 666 /dev/nst[0-9]           Change all tape devices.
chmod 666 /dev/sg7                Change a particular SCSI device.
```

Do not use **`chmod 666 /dev/sg[0-9]`** because this command provides direct access to anyone to all the SCSI devices on your system. Unless you always have the same devices that are turned on when you start Linux, you cannot safely predict the `sgN` number of a device.

Block counts

There is confusion about ISV zPDT that uses SCSI-attached tape drives that are defined as 3490 devices. This definition means that the actual device controls and sense information are transformed (by the awstape device manager) to appear as a 3490.

IBM 3490 tape drives provide a block counter. This counter is 22 bits, and the largest count that it can hold is approximately 4 million. If you write to a SCSI-attached tape drive that is defined as a 3490 (under zPDT), you receive an end-of-media indication after writing approximately 4 million blocks. z/OS performs EOV functions and calls for a new tape cartridge (depending on your JCL). The remaining tape media in the first cartridge is not used, but the system works correctly otherwise.

This situation is likely to arise only when using a SCSI 359x tape drive that is defined as a 3490 unit in the devmap. If you read a cartridge that is written by another system (that was not emulating a 3490 drive), the cartridge might contain more than 4 million blocks. This situation should work correctly if the IBM zSystems program does not read the block count. If it does, and if the tape position is past the 4 million block point, the IBM zSystems program indicates a block count error. What happens then depends on the design of the application program. A tape cartridge with more than approximately 4 million blocks is not fully compatible with 3490 emulation.

Parallel SCSI adapters

At the time of writing, the most recent Lenovo System x servers do not list parallel SCSI adapters for their standard configurations.

- IBM did not formally test any of the existing SCSI adapters with these servers.
- There is no known reason why they should not work if the appropriate adapter slots are available.

- We informally[11] used the Ultra SCSI 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 used openSUSE 11.2 or later for this operation. Other and earlier distribution with Linux kernels below the level that is 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, discuss your ISV zPDT configuration with your ISV 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 with differential circuits, low-voltage and high-voltage versions, and various terminators. If you are not familiar with this area, obtain expert help in configuring your system.
- The newest SCSI devices use fiber connections instead of parallel (wire) connections.

Testing specific hardware

Tip: Older SCSI drives might require interfaces that are not available with current servers. For example, some older drives require SCSI Fast/Wide High-Voltage Differential adapters. The last generation of these adapters used PCI (not PCIe) adapter slots in servers but many current servers no longer have these slots. The details that are involved in SCSI tape drive usage can be complex. Talk with a knowledgeable ISV zPDT provider about your SCSI tape drive requirements.

IBM testing (a few years ago) involved the following SCSI drives:

- IBM 3592-E05 TS1120 Fibre Channel-attached SCSI tape drive. The drive was at firmware level 1C91, as determined by the **`itdtinst1.2LinuxX86`** and **`itdtinst4.1.0.026LinuxX86_64`** tools from IBM.
- Fujitsu M2488E parallel SCSI tape drives with media cartridges that are compatible with IBM 3480 and 3490 drives. These drives were at firmware level 7.C.01 or 7.xG.01, as determined through the drives control panel.
- IBM LTO-3 3580 parallel SCSI tape drive. This drive was at firmware level 5BG4, as determined by the **`itdtinst1.2LinuxX86 tool`** from IBM.

These drives were tested with Lenovo System x servers x3650-M1 (7979), x3650-M2 (7947), and x3500-M2 (7939). The following adapters were used:

- Emulex Corporation Zephyr-X LightPulse Fibre Channel Host Adapter (rev 02). This adapter was at Emulex LightPulse x86 BIOS Version 1.71A0 and firmware ZS2.50A, as displayed during the BIOS startup.

  The Linux device drivers (provided in the Linux distribution) were determined by using the command **`dmesg | grep Emulex`**:

  – Red Hat Enterprise Linux (RHEL) 5.3 (64-bit): Emulex LightPluse Fibre Channel SCSI driver 8.2.0.33.3p

  – openSUSE 11.1 (64-bit): Emulex LightPluse Fibre Channel SCSI driver 8.2.8.7

  – SUSE Linux Enterprise Server 11 (64-bit): Emulex LightPluse Fibre Channel SCSI driver 8.2.8.14

[11] This statement was for ISV zPDT releases before GA10.

- QLogic Corporation ISP2432-based 4 Gb Fibre Channel to PCI Express HBA (rev 03). This HBA was at BIOS revision 1.28, as displayed during BIOS startup. The firmware level was 4.04.05 [IP][Multi-ID][84XX], as determined by the **ql-hba-snapshot.sh** tool from QLogic.

  The Linux device drivers (provided in the Linux distribution) were determined by using the command **dmesg | grep Qlogic**:

  - RHEL 5.3 (64-bit): QLogic Fibre Channel HBA Driver 8.02.00.06.05.03-k
  - openSUSE 11.1 (64-bit): QLogic Fibre Channel HBA Driver 8.02.01.02.11.0-k9
- Adaptec ASC-29320ALP U320 (rev 10) parallel SCSI adapter at Adaptec SCSI Card 29320LPE Flash BIOS v4.31.2S1, as displayed during BIOS startup.

Block size

You probably want to use variable block sizes with SCSI tape drives. You can check for them with these commands:

```
$ su                        (Switch to root. Might not be necessary.)
# mt -f/dev/st0 status
```

If the indicated tape block size is 0 bytes, the drive is set for variable block sizes. If not, enter the following command:

```
# mt -f/dev/st0 setblk 0
```

In our example, we use the `/dev/stN` devices rather than the `/dev/sgN` devices for these commands. Also, the Linux must have the **mt** command installed.

With our older drives and adapters, the Linux drivers for Emulex and QLogic both default to a maximum block size of 32 K. Block sizes larger than 32 K are typically not used by application programs, but large block sizes might be used by backup programs (such as ADRDSSU for z/OS) or by virtual tape managers. (The newer drivers that are used with the IBM 359x drives default to a 64 K maximum block size.)

The following Linux commands were used to set **def_reserved_size** to `65536`:

```
$ su                        (Changes to root.)
# rmmod sg                  (Removes the sg module.)
# /sbin/modprobe sg def_reserved_size=65536
      (Loads the sg module with the default reserved size up to 64 K.)
# cat /proc/scsi/sg/def_reserved_size
      (Displays the current setting for def_reserved_size.)
# exit                      (leave root)
```

This change allowed both cards to use 64 KB reserved buffer size for data transfers. These commands do not make a permanent change. A Linux restart puts the default size back to 32768.

Recent Linux releases might have changed these interfaces.

ISV zPDT SCSI utilities

Two ISV zPDT utilities can work directly with SCSI tape drives assuming that the Linux system has access to the SCSI adapter. Standard awstape-format files (in Linux) can be moved to and from SCSI tape devices by using the **scsi2tape** and **tape2scsi** commands:

```
$ scsi2tape /dev/st0 /z/my/TAPE23       (Copy SCSI tape to awstape file.)
$ tape2scsi /my/tapes/111111 /dev/st0   (Write SCSI tape from the awstape file.)
$ scsi2tape -c /dev/st0 /z/mytape2      (Compress the awstape file.)
```

These two commands are typically used when ISV zPDT is not active.[12]

Linux SCSI tape utilities

Linux can provide basic tape utility functions through the **`mt`** package. This package is not required for ISV zPDT, but it might be useful in other ways. The package is not normally installed by default. In some cases, the **`mt`** function is part of the **`cpio`** function.

Practical advice

Most SCSI tape drives do not use vacuum columns to help manage tape start and stop times. Instead, they use slower mechanical methods to manage physical tape movement, which means that starting and stopping the tape cannot be done in typical "tape gap" intervals.

For older types of drives, the net effect is usually as follows:

- If the program (including the operating system elements) issues tape read/write commands quickly enough, the tape drive runs the tape at full speed.
- If the program (including the operating system elements) does not issue reads/writes quickly enough, the tape drive stops after the current data block. The next read/write might cause the tape drive to "backhitch" (that is, back up the tape for a distance) and then restart forward movement. This action is done to ensure that the tape is "up to speed" for the next read/write operation.

This backhitch movement can greatly reduce the effective data speed of the drive. Not all drives encounter this situation because other factors such as internal buffering on newer types of drives can help avoid it. You can typically hear the effects of a tape drive doing a backhitch for every data block.

If you encounter this situation, an alternative approach is to use the ISV zPDT SCSI utilities such as **`scsi2tape`** to copy the SCSI tape to an awstape emulated tape volume. The SCSI utilities are fast and should not encounter any backhitching. Your application can process the emulated tape volume copy instead of the real SCSI tape volume, which might result in much faster processing of your tape data.

The SCSI adapters that are needed to use older SCSI tape drives might not be compatible with newer servers. One solution is to acquire an older server with appropriate adapters and use it to convert older tapes into awstape format files. This format can be readily moved to newer servers running ISV zPDT.

[12] If the SCSI devices are not in the active devmap, these utilities can be safely used while ISV zPDT is active.

15

Direct access storage device volume migration

The Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) package includes a client/server utility for moving 3390 volumes from a remote z/OS or z/VM system to ISV zPDT. The server portion of this utility runs under z/OS or z/VM on the remote IBM zSystems server. The remote server is typically a large IBM zSystems server, but it might be another ISV zPDT system. The client portion runs on the base Linux of your ISV zPDT machine.[1] The client and server are connected through TCP/IP. This utility is especially useful when transferring many volumes to an ISV zPDT system.

The server portion (on the remote z/OS or z/VM) reads all the tracks on a selected volume and sends them to the client (on the local base Linux). The client transforms it into the emulated 3390 format that is used by ISV zPDT and writes it as a Linux file. You use this file as an emulated volume under ISV zPDT.

There is no “reverse migration” function that is available through this client/server operation. If you must copy a 3390 volume from ISV zPDT to a “real” 3390, you can use the ADRDSSU program, as described in 12.8, “Moving 3390 volumes” on page 252.

One volume is processed for each client command that is sent to the server. You can create a Linux script with multiple invocations. The server portion (on z/OS) requires specific RACF authorizations. The server can copy active volumes, although the usefulness of the copy might be questionable depending on the volume activity at the time. The z/VM version requires that the server program has access to the full volumes being sent.

The speed of the copies depends on the TCP/IP bandwidth between the client and server, and the contents of each track. A considerable amount of data is involved on a typical 3390 volume; the transmission might take some time.

[1] ISV zPDT does not need to be operational while this utility is being used. Due to expected local area network (LAN) and disk activity, it is probably better to use the migration utility when ISV zPDT is not active.

A conceptual overview is shown in Figure 15-1.

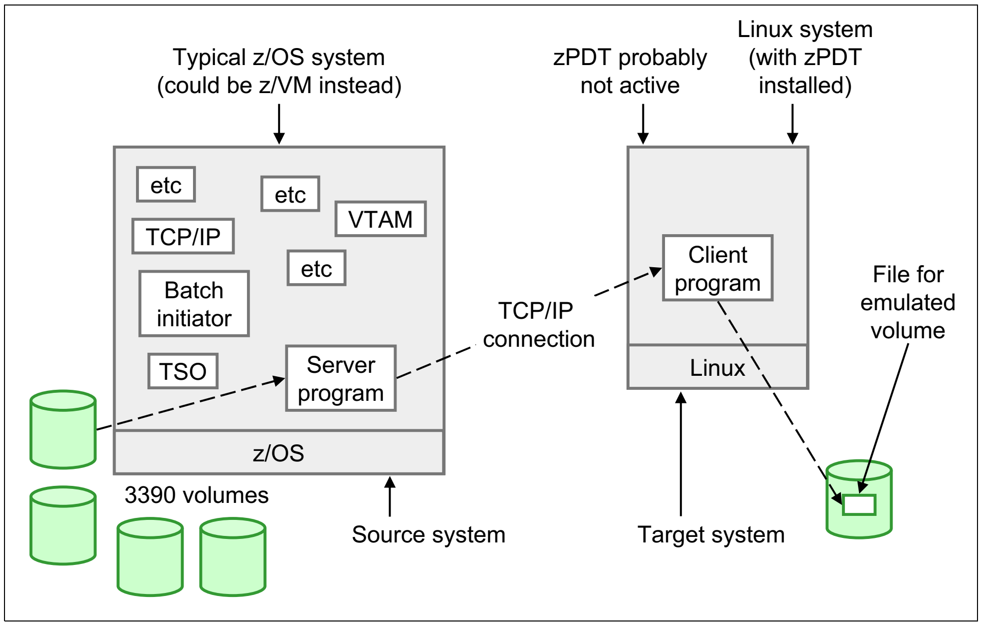


Figure 15-1 Volume migration overview (z/OS version)

15.1 Warnings

There are limitations on copying and using volumes from a z/OS system (and also from a z/VM system, with slightly different details). These limitations are not related to ISV zPDT or to the migration utility that is described in this chapter. A z/OS disk volume is not necessarily a stand-alone entity, depending on what is on the volume. If the volume contains Virtual Storage Access Method (VSAM) data sets of any type (including catalogs), then the volume contents are usable only if the complete VSAM metadata is available. Volume AAA might contain a VSAM data set that is cataloged on volume BBB. The VSAM information on volume AAA (including metadata in the VSAM Volume Data Set (VVDS)) must be synchronized with the catalog information on volume BBB. In this example, migrating (copying) both volumes might suffice if the catalog on volume BBB is later connected to the master catalog on the target system.

Volumes that do not contain VSAM data sets (or VVDS material) are safer to migrate in a stand-alone fashion. Migrating all the volumes of a z/OS system should be safe because this migration includes all the relevant catalogs and VSAM data volumes.

There is a special warning regarding the migration utility that is described in this chapter: No enqueue or serialization functions are used by this utility. Transferring an active volume is not a good idea. If the volume is active, it might have logical errors when the transferred copy is used. For example, the Volume Table of Contents (VTOC) might not reflect an additional extent that was allocated after the tracks containing the VTOC were copied, or the contents of a z/VM mini-disk on the volume might be changing as the associated tracks are copied.

With a z/VM system, the VM directory must be synchronized with any migrated direct access storage device volumes containing mini-disks.

15.2 Operational characteristics of the migration utility

The following characteristics of this utility are important:

- Complete volumes are transferred, which include initial program load (IPL) text, volume labels, VTOC, and *unallocated space*. The *logical* contents of the volume are not examined. Data sets on the volume are not recognized. The utility copies and transfers all the tracks on the volume, but it does not check whether the tracks are allocated (VTOC or virtual machine (VM) equivalents), in use (ENQ), or linked to a specific catalog (for VSAM, for example).
- A `read track` channel command word (CCW) is used to read the data on each track of a CKD volume. The amount of data on each track is variable, which means that the time to transmit a volume is variable, depending on how much data is on each track.
- Track data is not compressed for transmission.
- The source z/OS or z/VM (where the server component runs) is typically a large IBM zSystems server. The server can be an ISV zPDT system. The receiving Linux side must be a Linux system with ISV zPDT installed (but probably not active). The received copy of the source disk volume is stored in the awsckd (or awsfba if appropriate) format that is used by ISV zPDT.
- Specific RACF definitions are required for the source z/OS side. These definitions can protect the utility from misuse.
- The utility server program must be in an authorized library for z/OS.
- The utility server program is typically started as a batch job. It automatically terminates after 10 minutes of inactivity.
- The utility client program is run as a normal Linux command. It is possible to create a script file with multiple transfer commands so that multiple volumes can be transferred in an unattended manner.
- TCP/IP port 3990 is used by default.[2] The port number can be changed when starting the server and client.
- Both 3380 and 3390 volumes, any size, can be migrated, which include 3390 extended address volumes (EAVs) (“large volumes”).
- In practice, this utility is likely to run unattended because copying multiple volumes can take considerable time. Try a single volume and monitor the operation while it is running.
- Only one transfer can be active for the server. It is possible to run multiple servers in parallel (with different IP port numbers for each one), but the usefulness of this action is questionable.
- The z/OS version is only for the migration of CKD direct access storage device volumes. The z/VM version can migrate CKD and Fixed Block Architecture (FBA) direct access storage device volumes.
- As a best practice, do not use this utility while zPDT is in operation.

[2] Recent experience suggests that the default is not working. You should specify a port number; use port 3990 unless you have a reason to use a different port number.

15.3 Installing the migration utility for z/OS

Several steps are required to install the migration utility:

1. Upload the server module (ZOSSERV.XMIT) and install it in an authorized z/OS library. (The server module is in the /usr/z1090/bin directory that contains all the ISV zPDT executable files. It is an unloaded partitioned data set (PDS) member that was processed by the TSO **XMIT** command.)
2. Provide the required RACF definitions.
3. Determine whether TCP/IP port 3990 (on the z/OS side and the Linux side) were assigned for other purposes, and ensure that the port can pass through any firewalls.
4. Create a batch job to run the server.

15.3.1 Server installation

The /usr/z1090/bin/ZOSSERV.XMIT file must first be uploaded (in binary format) to a z/OS data set with dataset control block (DCB) characteristics RECFM=FB, LRECL=80, and BLKSIZE=3120. This XMIT file contains an unloaded PDS load library with one member. You can begin restoring this material by preallocating two data sets with the following job:

```
//OGDEN77 JOB 1,BILL,MSGCLASS=X
//  EXEC PGM=IEFBR14
//A DD DISP=(NEW,CATLG),UNIT=3390,VOL=SER=ZBSYS1,SPACE=(TRK,5),
//     DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120,DSORG=PS),
//     DSN=IBMUSER.SEQ.HOLDING
//B DD DISP=(NEW,CATLG),UNIT=3390,VOL=SER=ZBSYS1,SPACE=(TRK,(3,3,3)),
//     DCB=(LRECL=0,BLKSIZE=32760,RECFM=U),DSN=IBMUSER.PDS.HOLDING
```

The volser and data set names (DSN) are arbitrary. An experienced z/OS system programmer can handle this upload and installation in multiple ways. We present a basic example here. You can, for example, use IBM Interactive System Productivity Facility (ISPF) to create these two holding data sets.

After the receiving data set is ready, use binary mode and either IND$FILE[3] or FTP to transfer /usr/z1090/bin/ZOSSERV.XMIT to IBMUSER.SEQ.HOLDING. It is a relatively small file. Be certain to use a *binary transfer*.

After the module is uploaded to the sequential holding data set, issue the following Time Sharing Option (TSO) commands (by using the appropriate data set names):

```
RECEIVE INDATASET('IBMUSER.SEQ.HOLDING')
INMR901I DATASET ......
INMR906A Enter restore parameter or 'DELETE' or 'END' +
DATASET('IBMUSER.PDS.HOLDING')
INMR001I Restore successful ....
```

The name of the executable module after it is restored is ZPDTMSRV. You must select an authorized library (preferably in the LNKLST) to contain the server module. You must have the authority to update this library. We use USER.LINKLIB as an example of an authorized library in the LNKLST, but your system is probably different. User ID IBMUSER typically has update authority for all system libraries, and we use this user ID in our example. Again, your system might be different.

[3] IND$FILE is a z/OS program that works with the “file transfer” function that is present with many 3270 emulators.

Copy member ZPDTMSRV from your holding PDS to your authorized library. The easiest way to do this task is by using ISPF option 3.3. In the first panel of ISPF 3.3, make this selection:

```
Option=====> c
....
("from" data set name)
Name. . . . . . . 'IBMUSER.PDS.HOLDING'
(press Enter)
("to" data set name)
Name. . . . . . .'USER.LINKLIB'
(press Enter)
. ZPDTMSRV                            (Type over the period with the letter S and
press Enter.)
```

The server installation is complete. You can delete the holding data sets.

15.3.2 RACF requirements

Important: Consult with your RACF administrator for your z/OS system before making any RACF changes, especially if the DASDVOL class is active in the installation.

The server program *requires* RACF class DASDVOL to be active and the server program must have at least READ access to all volsers to be transferred. You must determine whether the DASDVOL class is active and used on the system where you install the migration utility server. You can do this task by logging on to TSO with a user ID with RACF SPECIAL authority and issue these TSO commands from a READY prompt or ISPF option 6:

```
SETROPTS LIST                     (Check the list of active classes.)
RLIST DASDVOL *                   (See whether any profiles are defined.)
```

If these checks are negative, you can assume that the DASDVOL class is not being used.

DASDVOL not in use

The next step is to decide whether only certain volumes should be subject to copying by this migration utility or whether all volumes might be accessed for migration. If you decide to allow migration of all volumes, enter these TSO commands:

```
SETROPTS CLASSACT(DASDVOL)          (Activate the DASDVOL class.)
SETROPTS RACLIST(DASDVOL)           (Optional, but recommended here.)
RDEFINE DASDVOL ** UACC(ALTER)      (Allow universal alter access.)
SETROPTS RACLIST(DASDVOL) REFRESH   (If you RACLISTed the class.)
```

The DASDVOL class is used only by a selected group of utility programs, such as dump or restore. Allowing **UACC(ALTER)** does not open all your data sets to access by all users. Whatever RACF data set protection that you have in place (through the **ADDSD** and **PERMIT** commands) is still effective.

This setup is the extent of the RACF setup if DASDVOL was not initially active and you want to allow the migration server to access any volume. If you want to limit the volumes that are subject to migration, you should work with an experienced RACF administrator. The general technique is to restrict global access to DASDVOL (perhaps with a **UACC(NONE)** condition) and then issue **PERMIT** commands to cover the volumes that you want to migrate. For example:

```
PERMIT VOL123 CLASS(DASDVOL) ID(IBMUSER) UACC(READ)
PERMIT ADCD* CLASS(DASDVOL) ID(IBMUSER) UACC(READ)
SETROPTS RACLIST(DASDVOL)REFRESH           (If you RACLISTed DASDVOL.)
```

In this example, volser VOL123 and any volser beginning with Application Development Controlled Distribution (ADCD) can be accessed by the migration utility.

The usage of the DASDVOL class might have side effects on other utility programs. If DASDVOL was not active before your migration activities, you might want to deactivate it when the migration activities are completed:

```
SETROPTS NORACLIST(DASDVOL)
SETROPTS NOCLASSACT(DASDVOL)
```

DASDVOL in use

If you find that the DASDVOL class is active on your server system, discuss the situation with the system programmers that manage that system. Your requirements are simple: you (meaning the user ID who runs the migration server program) need DASDVOL with READ access to whatever volsers that you intend to migrate.

TCP/IP port

By default, the migration programs use TCP/IP port 3990, but you can use a different port. You can look at the TCP/IP PROFILE on the z/OS system to see whether port 3990 is reserved for another application. However, another application can dynamically acquire the port. There is no easy way to prevent this situation, so specify a different port number only if there are error messages when you try to start the migration server or client. You also must verify that whatever firewalls are active allow port 3990 communication.

15.4 Operating the server under z/OS

The server should not be started until you are ready to use it. It automatically terminates after 10 minutes of inactivity. The following JCL can be used to start the server:

```
//MIGSERV JOB 1,OGDEN,MSGCLASS=X,TIME=1440
//ZPDTMIG EXEC PGM=ZPDTMSRV,REGION=0M,PARM='3990'
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTCPD DD DISP=SHR,DSN=ADCD.Z21Z.TCPPARM(TCPDATA)
```

Consider the following notes:

- The **TIME=1440** parameter is suggested because transmitting a full volume (or several volumes) can take considerable time. (It is much less than 1440 minutes.)
- The PARM field specifies the TCP/IP port number to use. Use port 3990 unless you have a specific reason to use a different port.
- The **SYSTCPD** statement must point to the **DATA** statement that is used by the z/OS TCP/IP stack. You must determine the correct data set name for your z/OS system. One way to do this task is to examine the procedure that is used to start TCP/IP on your system.
- If your authorized library is not in LNKLST, you need a **JOBLIB** or **STEPLIB** statement that references only authorized libraries.
- The user submitting this job must have a user ID that has at least READ access to the appropriate volsers in DASDVOL.

15.5 Installing the server under z/VM

The z/VM system must be Release 5.4 or later.

The `ZVMSERV.XMIT` file is with other ISV zPDT binary material in `/usr/z1090/bin` on the ISV zPDT system. This file must be uploaded in `BINARY, FIXED LRECL 80` format to a Conversational Monitor System (CMS) user's A disk. (This file essentially contains a card deck.)

After this task is complete, complete the following steps from CMS:

1. Issue the **`CP SPOOL PUN`** command.
2. Issue the **`PUNCH ZVMSERV XMIT A (NOH`** command.
3. Issue the **`RDRLIST`** command.
4. Next to the file in your `RDRLIST`, issue the **`RECEIVE`** command.

The results are a reconstructed executable file on your CMS A disk, which completes the server installation.

15.6 Operating the server under z/VM

Your z/VM must have TCP/IP active and connected to the appropriate network. The CMS user running the migration utility must have access to the volumes to be migrated.

The CMS user starts the utility by issuing a **`ZVMSERV`** command. A single operand specifies the TCP/IP port number, which defaults to port 3990. When started, the server should indicate that it is waiting for a client connection. The server times out after 10 minutes without a client connection.

If a z/VM mini-disk is migrated, it appears as a small 3390 volume on the receiving system. It is not a mini-disk on a larger z/VM volume anymore.

15.7 The client commands

There are two client commands:

- **`hckd2ckd`**: Used with both z/OS and z/VM to migrate a CKD direct access storage device volume.
- **`hfba2fba`**: Used only with z/VM to migrate an FBA direct access storage device volume.

Important: Some newer ISV zPDT users with extensive personal computer (PC) experience (and less IBM zSystems experience) misunderstand the meaning of FBA here. It refers to an older IBM zSystems disk architecture that is becoming rather rare. z/OS, for example, does not use it. In the context of this publication and ISV zPDT usage, it does *not* refer to PC (or UNIX or Linux) style physical disks.

This general syntax of the client commands (entered on the Linux client machine by using a Linux command-line interface (CLI)) is as follows:

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

- **host** is the TCP/IP name of the system with the matching server program, which might be a dotted decimal address or a name that can be resolved by Linux TCP/IP.
- **:port** is a TCP/IP port number that is used by both the client and server programs. It defaults to 3990. Our recent experience indicates that the port number does not default correctly, so it is a best practice to include the port number in your command.
- **outfile** is a file name (on the Linux) system where the migrated volume is placed (in awsckd, awsfba, or awstape format).
- **-v** or **--volser** indicates the 3380/3390 volume (on the remote z/OS system) that will be copied (migrated).
- **-u** or **--unit** indicates the address (device number) of the volume that is to be copied (migrated).

Either the **-u** or **-v** parameter must be supplied for direct access storage device, but not both. The **-u** parameter is normally used for tapes.

After the server is started on the z/OS or z/VM system, the client can be started on the Linux system. ISV zPDT does not need to be operational for this task. (As a best practice, it should not be operational because the migration utility can place a heavy load on the LAN interface.) Examples of commands that can be used to run the client are as follows:

- **`$ hckd2ckd 192.168.1.99:3990 /z/VOL123  -v VOL123`**
- **`$ hckd2ckd BIG.ZOS.ADDR:3990 /z/VOL678 -u A8F`**
- **`$ hckd2ckd 192.168.1.99:3990 /z/host.WORK23 -v WORK23 --norestart`**

The first operand is the IP address of the z/OS system where the server is running. The TCP/IP port number can be changed as shown in the examples, where we use port 3990. (The server must be started by using the same port number.) The second operand is the Linux file name that is used to store the migrated volume. Either the **-v** or **-u** parameter must be specified. The **-v** parameter is a volser and the **-u** parameter is a device address (device number) on the server system. These parameters determine which volume is processed.

15.8 Additional notes

After a volume is migrated to Linux, you can add it to your device map (devmap) and access it from ISV zPDT. You should check the permission bits for the file. (The ISV zPDT system must have read/write access to it.) Our examples are for z/OS volumes, but the volume can be for z/VM or another operating system.

Linux volumes

We discovered an interesting situation when migrating Linux for IBM zSystems volumes. Our particular experience was with SUSE Linux Enterprise Server-10 SP1 for IBM zSystems, but it might apply to other distributions.

With many Linux distributions, several *fstab options* can be selected for controlling disk volume mounts. These options include mounting by device name, volume label, UUID, device ID, or device path. The default is often to mount by device ID. This action produces a boot parameter list (and fstab) like the following one:

```
parameters='root=/dev/disk/by-id/ccw-IBM.750000000M1881.2c23.1c-part1'
```

This disk identification is unique to the original disk drive and useless when the volume is copied or migrated to another disk. In this situation, the migrated Linux volumes could not be started. This identification is best changed when installing Linux by selecting a volume label (for example, `LABEL=rootfs`) or device name (for example, `/dev/dasda1`) when initially creating the disk partitions. The naming can be changed later by carefully editing `/etc/zipl.conf` and `/etc/fstab`. In any event, the naming should be changed before migrating the volumes. Unfortunately, it is not always obvious how to select which disk naming convention should be used when installing Linux.

Multiple TCP/IP stacks

A z/OS system with multiple TCP/IP stacks presents an additional complication. In this situation, an extra step is needed in the server job:

```
//MIGSERV JOB 1,OGDEN,MSGCLASS=X,TIME=1440
//STEP0   EXEC PGM=BPXTCAFF,PARM='TCP342'
//*
//ZPDTMIG EXEC PGM=ZPDTMSRV,REGION=0M,PARM='3990'
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTCPD DD DISP=SHR,DSN=ADCD.xxxx.TCPPARM(TCPDATA)
```

The **BPXTCAFF** program is used to associate a specific TCP/IP stack with the current address space. The **PARM** value is the job name that is used to start the TCP/IP stack.

Typical client usage

A typical usage of the client might be as follows:

```
$ hckd2ckd 192.168.1.81:3990 /z/ZBRES1 -v ZBRES1
AWSHTC090I Host name   : 192.168.1.81:3990
AWSHTC091I     Restart : No
AWSHTC095I     Vol-Ser : ZBRES1
AWSHTC096I      Output : /z/ZBRES1
AWSHTC097I Transferring 3990 volume of 3339 cylinders
AWSHTC098I Cylinder nnnn ...
```

We found that transferring a fairly full 3390-3 volume on a private 100 Mbps LAN took approximately 11 minutes. This process consumed roughly 150 processor seconds on the source z/OS system (which was a different ISV zPDT system on a moderate-performance PC).

When the migration completes, the cylinder number that is displayed is one less than the actual number of cylinders that are transferred. Also, there might be a delay (up to 30 seconds) between the last message and the time that the command ends. Both these conditions are normal.

Migrating a list of volumes

Using **gedit** or another editor, you can create a Linux file that is named `mig`, for example. The `mig` files contain this information:

```
hckd2ckd 192.168.1.81:3990 /z/ZOS111/ZBRES1 -v ZBRES1
hckd2ckd 192.168.9.01:3990 /z/ZOS111/ZBRES2 -v ZBRES2
hckd2ckd 192.168.9.01:3990 /z/ZOS111/ZBSYS1 -v ZBSYS1
hckd2ckd 192.168.9.01:3990 /z/ZOS111/ZBUSS1 -v ZBUSS1
hckd2ckd 192.168.9.01:3990 /z/ZOS111/backup/MYVOL1xx -v MYVOL1
```

You can start the server program on the z/OS host and then run the commands that are in your `.mig` file:

```
$ ./mig                      (Run the commands in file that is named mig.)
```

This command transfers all the volumes that are listed without any further manual intervention.

16

Channel-to-channel

The awsctc device manager provides channel-to-channel (CTC) functions by using TCP/IP communication paths. As shown in Figure 16-1, the connection can be within the same Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) instance, between ISV zPDT instances on the same machine, or between ISV zPDT instances on different machines. Among other functions, CTC can be used by z/OS for global resource serialization (GRS) “rings”, Network Job Entry (NJE) connections, Cross-System Coupling Faciity (XCF), TCP/IP connections, and so forth. Typically, the usage is between separate personal computer (PC) machines.

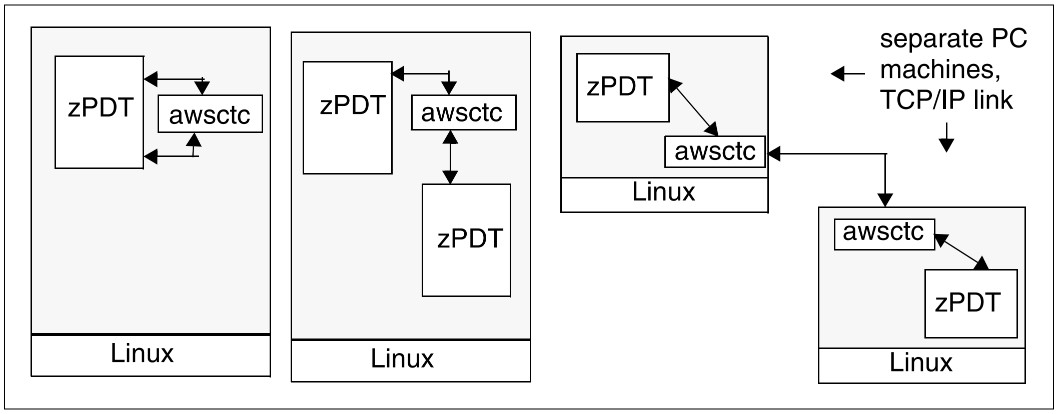


Figure 16-1 CTC links for ISV zPDT

The awsctc device manager emulates IBM 3088-4 (or 3088-8) control units and devices.[1] The 3088 is an older product that provided the “middle” function for connecting two channels to each other. This control unit function is integrated with the channels into modern systems, and physical 3088 boxes are no longer used. However, the logical function has not changed.

[1] These units were parallel channel control units. ESCON and FICON CTC operation are not emulated.

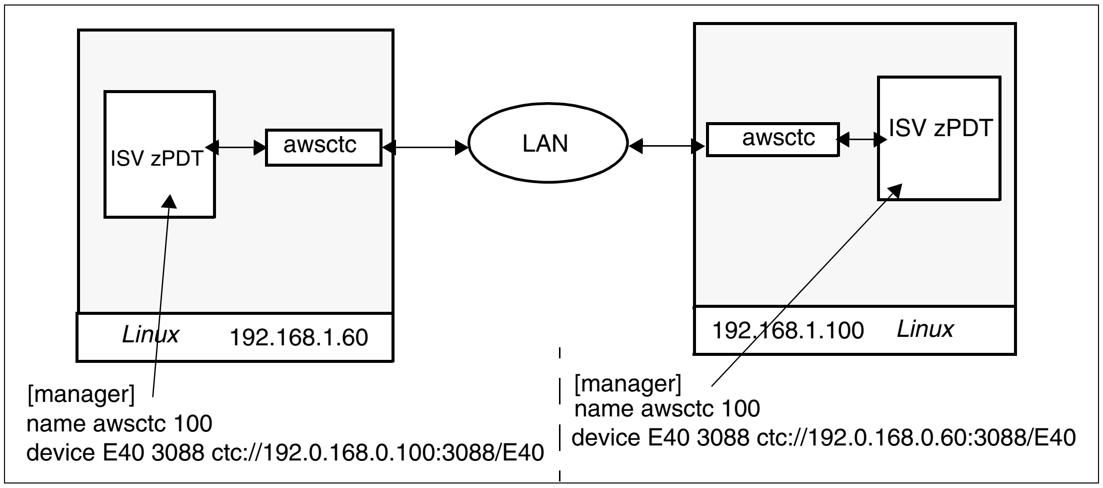
The device map (devmap) stanza for awsctc appears as follows:

```
[manager]
name awsctc 75
device E40 3088 3088 ctc://otherhost:3088/E42
device E43 3088 3088 ctc://192.168.1.90:3089/E43
```

The last parameter of the device statement contains three elements:

- A TCP/IP address (in dotted decimal form or as a name that can be resolved by Linux).
- Following a colon, a TCP/IP port number that is to be used on both the local and other system. The same port number is used on both ends of the connection.[2] If multiple CTC connections are defined, a different port number must be specified for each connection.
- Following a slash, the device number (address) of the corresponding CTC device on the remote system. To specify this number, you must know how the devmap is defined on the other system. The same device number is often used at both ends, but it is not required. The numbers should correspond to CTC devices in the z/OS I/O definitions that are being used.

An example of a simple connection between two ISV zPDT instances in two different machines is shown in Figure 16-2.



```
[manager]
name awsctc 100
device E40 3088 ctc://192.0.168.0.100:3088/E40
```

```
[manager]
name awsctc 100
device E40 3088 ctc://192.0.168.0.60:3088/E40
```

Figure 16-2 Simple two-system CTC definition

In this example, two separate ISV zPDT instances are shown. Both instances use the same device number (E40) for the CTC device, although there is no requirement to do so. Both ends of the connection *must* specify the same port number (3088 in the example). Each definition specifies the IP address of the other base Linux system.

Devmap notes

Although `ctc://` is optional, if a protocol is specified, it must be `ctc`. The port numbers below 1024 cannot be used, so the port number must be in the range of 1024 - 65535. The IP address string cannot be specified with an **`awsmount`** command.

Each CTC address must be defined. An example 3088 had multiple device numbers that were defined in blocks of 8, 16, 32, or 64 devices. For example, a 3088-4 had 16 devices starting at a specified address. The emulated CTC does not use this setup. Each specific device number must be defined.

[2] We use ports 3088 and 3089 for these examples because they are easy to remember. There is nothing special about these port numbers.

Status display

You can use the **awsstat** command to display the status of the CTC device. When the device's peer is not available, the status flips between `connecting` and `accepting`. After the peer is available, the status changes to either `connected` or `accepted`. The `accepted` side is considered the A side for protocol conflict resolution. This status does not make any difference to the user. After data begins to flow, `accepted/connected` is shortened to `A` or `C` and the number of send/receive bytes displays.

16.1 ISV z/OS usage example

Configurations that use CTC can be complex. Here is a basic NJE example, as shown in Figure 16-3.

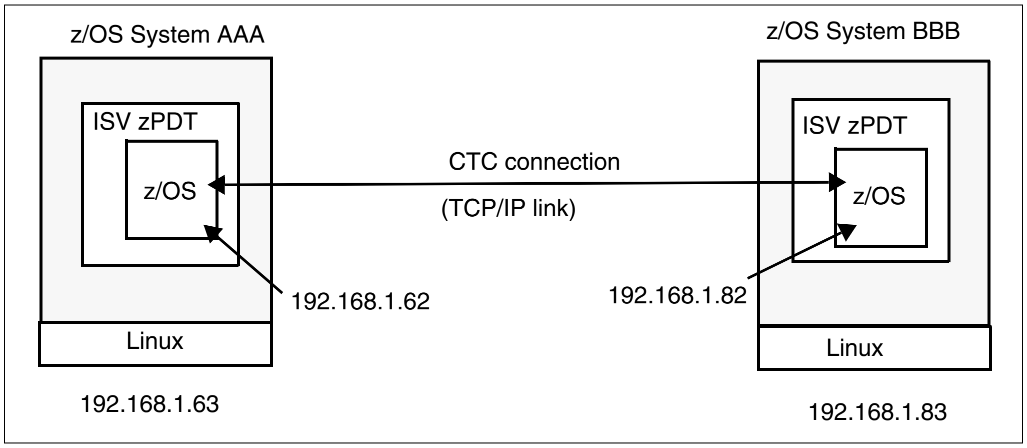


Figure 16-3 Trivial NJE setup

The JES2PARM data for system AAA might include the following lines:

```
NJEDEF DELAY=360,LINENUM=2,JRNUM=2,JTNUM=2,NODENUM=2,OWNNODE=1,
       PATH=1,RESTTOL=100,RESTMAX=8000000,RESTNODE=100,SRNUM=2,
       STNUM=2,MAILMSG=YES,TIMETOL=1440
LINE(1) UNIT=E40,TRANSPAR=YES
NODE(1) NAME=AAA,PATHMGR=NO
NODE(2) NAME=BBB,PATHMGR=NO
CONNECT NODEA=AAA,NODEB=BBB
```

The JESPARM data for system BBB might include the following lines:

```
NJEDEF DELAY=360,LINENUM=2,JRNUM=2,JTNUM=2,NODENUM=2,OWNNODE=2,
       PATH=1,RESTTOL=100,RESTMAX=8000000,RESTNODE=100,SRNUM=2,
       STNUM=2,MAILMSG=YES,TIMETOL=1440
LINE1 UNIT=E40,TRANSPAR=YES
NODE(1) NAME=AAA,PATHMGR=NO
NODE(2) NAME=BBB,PATHMGR=NO
CONNECT NODEA=AAA,NODEB=BBB
```

These examples assume that device number E40 is defined in z/OS as a CTC device. (This situation is true for current z/OS Application Development Controlled Distribution (ADCD) systems.) Use the IP address of the remote Linux system (and not the remote z/OS system) in the devmaps.

Important: The time-of-day (TOD) on the two systems (in this example) must be within a few seconds of each other or the CTC connection cannot be opened.

The devmap for system AAA might contain the following stanza:

```
[manager]
name awsctc 300
device E40 3088 3088 ctc://192.168.1.83:3088/E40
```

The devmap for system BBB might contain the following stanza:

```
[manager]
name awsctc 300
device E40 3088 3088 ctc://192.168.1.63:3088/E40
```

The status of the CTC connection can be monitored with the **`awsstat`** command:

```
$ awsstat E40                (Use the correct device number.)
```

The device status usually starts as `connecting` or `accepting`, and later switches to `connected` or `accepted`. This task might take some time. After the connection is accepted and used, the status displays byte counts, such as `C-160/162`.

We found that the best approach is to wait until the CTC link is established before performing an initial program load (IPL) of z/OS. The link can be checked with the **`awsstat E40`** command (by using the correct device number for your CTC). This task normally takes only a few seconds, but can take longer in rare cases. Then, issue commands on each Job Entry Subsystem (JES) 2 to make the connection operational:

```
$S N,LINE1                   (Sometimes needed. We needed it.)
$S LINE1
```

A job that is submitted on system AAA can be directed to system BBB for running by the following JCL:

```
//MYJOB JOB 1,OGDEN,MSGCLASS=X,USER=IBMUSER
//  XMIT DEST=BBB
//MYJOB2 JOB 1,OGDEN.MSGCLASS=X,USER=IBMUSER
//STEP1 EXEC PGM=IEFBR14
```

Notice the two **`JOB`** statements in this example. The first is needed to “introduce” the **`XMIT`** statement into the job stream. The **`XMIT`** statement sends everything after it to the indicated node, including the second **`JOB`** statement. The first **`JOB`** statement (before the **`XMIT`**) is not sent to the remote node.

The following JES2 commands might be useful when working with this NJE connection:

```
$DCONNECT                    (useful status display)
$S LINE(1)
$S N,LINE1                   (Discover dynamic connections.)
$D NODE(*)
$D LINE
$D DESTID(*)
$D PATH(*)
$P LINE(1)                   (stop line, which is an important command)
$E LINE(1)                   (Reset line.)
$D NJEDEF                    (Display NJE definitions.)
$N,D=BBB,'$D NJEDEF'         (Send a JES2 command to the other node.)
```

A JES2 cold start might be necessary to enable NJE changes to JES2PARM. You must stop the line `$P LINE(1)` when stopping JES2. If the `$P` is not effective, try `$E LINE(1)` followed by `$P LINE(1)`.

When stopping JES2 activity (to prepare for a z/OS shutdown, for example), any started NJE lines must be stopped or JES2 does not end.

Tip: If a CTC operation is defined in the devmap, as shown in this example, but the TCP/IP connection fails or does not start, there will be periodic warning messages from ISV zPDT. If the situation exists for an extended period, there can be an excessive number of these messages.

16.2 Multiple instances and z/VM

The following material outlines several uses of CTC connections with two instances of z/VM.[3] This material is intended as a general overview and does not provide complete step-by-step instructions for implementation and usage. It is unlikely that all the links that are shown in this example will be present in any practical system. The details that are shown here are from an older ISV zPDT and z/VM. A more modern configuration has not been tried.

16.2.1 Devmaps

Three devmaps are needed for these examples. One is for a group controller (for shared devices), and two are for the z/VM instances. Many of the choices are arbitrary.

Group controller (ibmgroup)

```
[system]
members ibmsys1 ibmsys2
3270port 3270

[manager]
name aws3274 0002
device 0020 3279 3274
device 0021 3279 3274     # More devices could be included.

[manager]
name awsosa 0009 --path=A0 --pathtype=OSD --tunnel_intf=y
device 0E00 osa osa
device 0E01 osa osa
device 0E02 osa osa
```

ibmsys1 instance

```
[system]
memory 1024m
processors 1
group ibmgroup

[manager]
name awsckd 0001
device 1000 3390 3990 /z1/540res     #unshared disks
```

[3] Thanks to Bruce Hayden of the Washington Systems Center for this material.

```
device 1001 3390 3990 /z1/540spl
device 1002 3390 3990 /z1/540pag
device 1003 3390 3990 /z1/540w01
device 1004 3390 3990 /z1/540w02

[manager]
name awsctc 0075
device 700 3088 3088 ctc://localhost:3700/700   #for TCP/IP
device 700 3088 3088 ctc://localhost:3701/701   #for TCP/IP
device 710 3088 3088 ctc://localhost:3710/710   #for TSAF
device 720 3088 3088 ctc://localhost:3720/720   #for ISLINK
device 740 3088 3088 ctc://localhost:3740/740   #for RSCS
```

ibmsys2 instance

```
[system]
memory 1024m
processors 1
group ibmgroup

[manager]
name awsckd 0001
device 1000 3390 3990 /z2/540res      #unshared disks
device 1001 3390 3990 /z2/540spl
device 1002 3390 3990 /z2/540pag
device 1003 3390 3990 /z2/540w01
device 1004 3390 3990 /z2/540w02

[manager]
name awsctc 0075
device 700 3088 3088 ctc://localhost:3700/700   #for TCP/IP
device 700 3088 3088 ctc://localhost:3701/701   #for TCP/IP
device 710 3088 3088 ctc://localhost:3710/710   #for TSAF
device 720 3088 3088 ctc://localhost:3720/720   #for ISLINK
device 740 3088 3088 ctc://localhost:3740/740   #for RSCS
```

Description

Each instance has its own copy of the z/VM disks, and they are in separate directories (`/z1` and `/z2`).

ISLINK

An **`ISLINK`** creates an Inter-System Facility for Communications (ISFC) connection. It is the easiest link to use because it involves only CP commands and does not require a virtual machine (VM) or user ID. The VM system identifiers of the two connected systems must be different. Using the devmaps that are shown above, the following commands (issued by `MAINT`, for example) activate the link:

```
CP ACTIVATE ISLINK 0720           (command on ibmsys1)
CP ACTIVATE ISLINK 0720           (command on ibmsys2)
```

This command should result in the message `HCPALN2702I Link 0720 came up`. The command **`Q ISLINK`** can be used to display the status of the connection.

Transparent Services Access Facility

Transparent Services Access Facility (TSAF) is an older function and that is not used if ISFC is available. To try it, use commands such as the following ones (issued by MAINT on both systems):

```
CP XAUTOLOG TSAFVM
CP ATT 0710 TSAFVM
```

Then, log on to TSAFVM and enter **ADD LINK 0710** on both sides. The status of the links can be displayed with **Q LINKS ALL**.

Remote Spooling Communications Subsystem

Remote Spooling Communications Subsystem (RSCS) can be more complicated. The RSCS product is not enabled by default. You can check its status with the command **Q PRODUCT STATE ENABLED**. If RSCS is not in the list, it can be enabled (with user **MAINT**) by using the command **SERVICE RSCS ENABLE** followed by **PUT2PROD**.

A configuration file is necessary on each system. The configuration files are placed on the RSCS 191 disk with the name RSCSTCP CONFIG. An example for each system might be as follows:

For ibmsys1

```
LOCAL   IBMSYS1       *  RSCS
LINKDEFINE IBMSYS2 TYPE NJE LINE 740 QUEUE PRI NODE IBMSYS2
PARM IBMSYS2  STREAMS=2 MAXU=2 MAXD=10 LISTPROC=NO TA=1 TAPARM='TH=100'

AUTH  *  OPERATOR *   CP
AUTH  *  MAINT    *   CP
```

For ibmsys2

```
LOCAL   IBMSYS2       *  RSCS
LINKDEFINE IBMSYS1 TYPE NJE LINE 740 QUEUE PRI NODE IBMSYS1
PARM IBMSYS1  STREAMS=2 MAXU=2 MAXD=10 LISTPROC=NO TA=1 TAPARM='TH=100'

AUTH  *  OPERATOR *   CP
AUTH  *  MAINT    *   CP
```

After the configuration files are available, start RSCS with the command **XAUTOLOG GCS** (issued on both systems). After RSCS starts, issue the command **SMSG RSCS START IBMSYS1** on ibmsys2 and **SMSG RSCS START IBMSYS2** on ibmsys1. The status of the RSCS connection can be displayed with **SMSG RSCS Q SY**.

TCP/IP

TCP/IP requires a pair of CTC links, one for read and one for write. The easiest way to set up TCP/IP for z/VM is with **IPWIZARD**. With **IPWIZARD**, you must assign hostnames and domain names. In an isolated environment, these names can be arbitrary. For an isolated environment with only two nodes, the gateway address does not matter, so you might use 10.1.1.1.[4] Use CTC0 as the interface name and address 0700 (from our sample devmap). We assigned IP address 10.1.1.2 to ibmsys1 and 10.1.1.3 to ibmsys2, and used a mask of 255.255.255.0. The configuration dialog includes positional parameters to indicate which device to use as the read channel and which to use as the write channel. This parameter could be 0 on ibmsys1 and 1 on ibmsys2.

[4] The default address of the tunnel connection to the base Linux is 10.1.1.1. This default is used in the sample devmap.

The status of TCP/IP can be checked by running the following commands:

```
VMLINK TCPMAINT 592
NETSTAT DEVL
```

You can force TCP/IP to restart by running the following commands:

```
FORCE TCPIP
XAUTOLOG TCPIP
```

Cryptographic usage

An Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) system can emulate multiple cryptographic adapters as CEX8S devices.[1] Each CEX8S emulated adapter runs as separate Linux processes. If sufficient base processor cores are available so that these threads can be dispatched in parallel by Linux, the emulated adapters can run asynchronously with the ISV zPDT CPs.

Do not confuse a cryptographic *adapter* (often known as a *co-processor*) with the cryptographic *instructions* that are always available with an ISV zPDT system. These instructions are the KM, KMC, KIMD, KLMD, KMAC, and associated instructions (known as the CP Assist for Cryptographic Functions (CPACF) instructions) that provide several fundamental cryptographic operations.

The cryptographic instructions can be coded directly in a program or used through Integrated Cryptographic Service Facility (ICSF) programming interfaces. For practical purposes, the cryptographic co-processor facilities are available only through ICSF programming interfaces.

Important: The cryptographic adapter emulation that is used with ISV zPDT GA11 uses a new format for storing master keys. ISV zPDT GA11 does not automatically convert an earlier format to the current format. You must initialize the GA11 cryptographic adapter with the same master keys previously used if you need to access previously (before GA11) encrypted data.

When attempting to initialize ISV zPDT GA11 emulated cryptographic adapter master keys with the same keys that were used with earlier versions (by typing them into the z/SO @ICSF program), it is important to enter the master key *exactly* as it was entered for the previous system.

For ISV zPDT GA11, do not copy the previous `~/z1090/srdis` files to the new GA11 system because they are not compatible with GA11. Furthermore, if you might use your older ISV zPDT version again, be certain to save the older `srdis` file (under a different name) before creating one with ISV zPDT GA11.

[1] This situation assumes that you use ISV zPDT GA11 or later. Earlier releases of ISV zPDT had earlier levels of CEX functions.

17.1 Background information

A cryptographic co-processor is represented by an adjunct processor (AP) process in ISV zPDT. An IBM zSystems CP sends and receives work entries through co-processor queues that are known as *domains*.

The *accelerator mode and customized (user-defined extensions (UDXs))* functions of a cryptographic co-processor are not provided for ISV zPDT. Trusted Key Entry (TKE) systems, which are personal computers (PCs) with unique software and an appropriate cryptographic adapter, are not used with ISV zPDT. PKCS#11 is not available with ISV zPDT emulation.

Each emulated cryptographic co-processor has 16 domains.[2] Each z/OS instance uses a different domain (as specified in the ICSF parameters). If multiple co-processors are defined, then z/OS uses the same domain number in each co-processor. If multiple co-processors are defined, z/OS can dispatch requests to all of them (by using the same domain number in each processor).

With ISV zPDT, the cryptographic co-processor keys (and other state information) are stored in the `~/z1090/srdis` directory. There is a separate subdirectory for each defined co-processor. These directories are created automatically by ISV zPDT if they do not exist. Any tampering with this information can produce unpredictable results. However, a complete co-processor subdirectory can be deleted as a way of reinitializing (zeroing) that co-processor. (The **`ap_zeroize`** command, described later, is another way to re-initialize a co-processor.)

ISV zPDT cryptographic functions are intended for development purposes and not for security. Although the format of the key data in `~/z1090/srdis` is not documented, it is not secure in any cryptographic sense.

The provided ISV zPDT cryptographic functions should be functional for normal, basic uses. ISV zPDT is not intended for more exotic cryptographic enhancements.

17.2 Device map specification

The ISV zPDT specification for emulated cryptographic adapters is part of the device map (devmap) and consists only of a simple `adjunct-processors` stanza:

```
[system]
memory 8000m
3270port 3270
processors 2

[adjunct-processors]
crypto 0
crypto 1                    #(More than one cryptographic adapter is possible.)

[manager]
name .........
```

With the ISV zPDT architecture, you can have up to 16 or 64 cryptographic co-processors, depending on the overall configuration.[3] z/OS allows a maximum of 16.

[2] CEX8S functions on a larger IBM zSystems server have more domains. ISV zPDT is limited to 16 domains.

[3] An ISV zPDT group controller instance can have up to 64 co-processors defined. Otherwise, the limit is 16.

17.3 Initial ICSF startup

For practical purposes, the ICSF software functions are needed to initialize cryptographic adapters. Here is a brief outline of the steps that we took to customize and use the ICSF panels on a z/OS Application Development Controlled Distribution (ADCD) 2.4 system. The usage of the ADCD system `PARMLIB` and `PROCLIB` is not required, so adjust these names for your needs.

Recent ADCD z/OS releases automatically start Cryptographic Service Facility (CSF) and have the basic customization to use both the CPACF instructions (which are always enabled) and CEX8S functions (if defined in your devmap). The most recent z/OS ADCD (at the time of writing) uses ICSF level HCR77D2.

We found that we needed to specify a domain number in the `PARMLIB` member that is associated with ISCF:

- For `PARMLIB` member `CSFPRM00` (this example is from a recent ADCD):

```
CKDSN(CSF.CSFCKDS)       <-- You might want to change these data set names.
PKDSN(CSF.CSFPKDS)           (Perhaps with an “S” before the “CSF...”)
COMPAT(NO)
DOMAIN(0) <-- Add this parameter to specify a domain number.
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

- For `PARMLIB` member `IKJTSO00`:

```
AUTHPGM NAMES(         /* AUTHORIZED PROGRAM NAMES     */ +
                                ...
CSFDAUTH        /*THIS WAS ALREADY IN OUR ADCD SYSTEM */ +
CSFDPKDS       /*WE ADDED THIS ONE                    */ +

AUTHTSF NAMES     /*PROGRAMS......                    */ +
....
CSFDAUTH        /*THIS WAS ALREADY IN OUR ADCD SYSTEM */ +
CSFDPKDS       /*WE ADDED THIS ONE                    */ +
```

Be certain to copy the plus signs (+) at the end of each line. These additions should prevent `047 ABEND` errors when customizing the cryptographic co-processor.[4] After making the `PARMLIB` changes, stop CSF (**`P CSF`**) and restart it (**`S CSF`**).

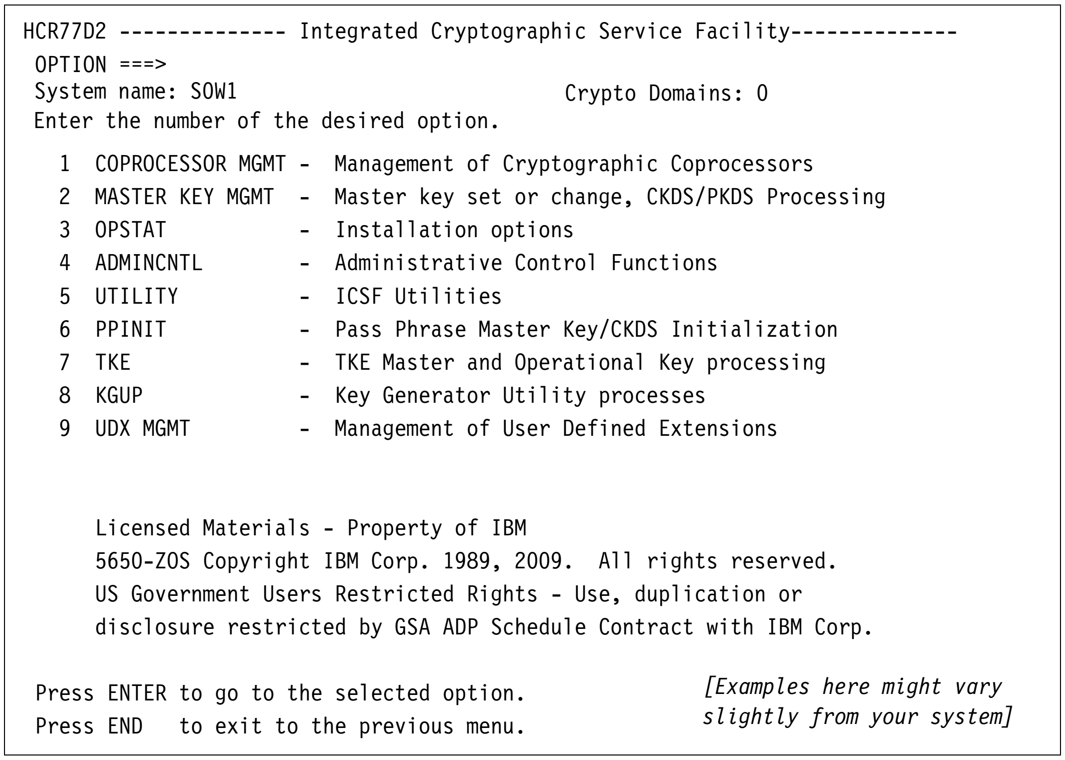
If you added names to the `IKJTSO00` member, you might want to perform an initial program load (IPL) of your z/OS system to prevent an `047 ABEND` when initializing ICSF.[5] An alternative is to try a **`set ikjtso00`** command (assuming that you used member “00.”)

[4] Later ADCD releases might not need these additions.

[5] Logging off Time Sharing Option (TSO) and then logging on again might accomplish this task.

Complete the following steps:

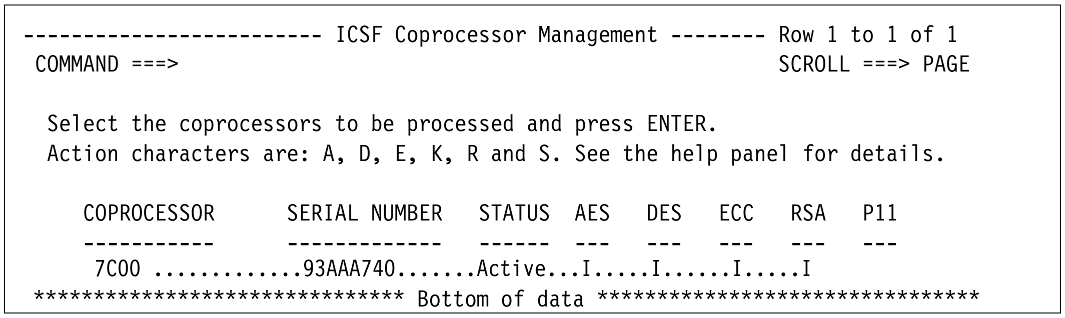
1. To begin customization of a cryptographic co-processor, go to IBM Interactive System Productivity Facility (ISPF) option 6 and enter the command **@ICSF**. This command should produce the first ICSF panel, as shown in Figure 17-1. (The panel examples are from an older z/OS system.)



```
HCR77D2 -------------- Integrated Cryptographic Service Facility--------------
 OPTION ===>
 System name: SOW1                             Crypto Domains: 0
 Enter the number of the desired option.

   1  COPROCESSOR MGMT -  Management of Cryptographic Coprocessors
   2  MASTER KEY MGMT  -  Master key set or change, CKDS/PKDS Processing
   3  OPSTAT           -  Installation options
   4  ADMINCNTL        -  Administrative Control Functions
   5  UTILITY          -  ICSF Utilities
   6  PPINIT           -  Pass Phrase Master Key/CKDS Initialization
   7  TKE              -  TKE Master and Operational Key processing
   8  KGUP             -  Key Generator Utility processes
   9  UDX MGMT         -  Management of User Defined Extensions



      Licensed Materials - Property of IBM
      5650-ZOS Copyright IBM Corp. 1989, 2009.  All rights reserved.
      US Government Users Restricted Rights - Use, duplication or
      disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

 Press ENTER to go to the selected option.          [Examples here might vary
 Press END   to exit to the previous menu.           slightly from your system]
```

Figure 17-1 First ICSF panel

2. On this panel, select option 1. This action should produce a display like Figure 17-2. This panel verifies that the co-processor is active. The “7C00” nomenclature varies with releases.



```
------------------------- ICSF Coprocessor Management -------- Row 1 to 1 of 1
 COMMAND ===>                                                  SCROLL ===> PAGE

  Select the coprocessors to be processed and press ENTER.
  Action characters are: A, D, E, K, R and S. See the help panel for details.

     COPROCESSOR      SERIAL NUMBER   STATUS  AES   DES   ECC   RSA   P11
     -----------      -------------   ------  ---   ---   ---   ---   ---
      7C00 .............93AAA740.......Active...I.....I......I.....I
******************************* Bottom of data ********************************
```

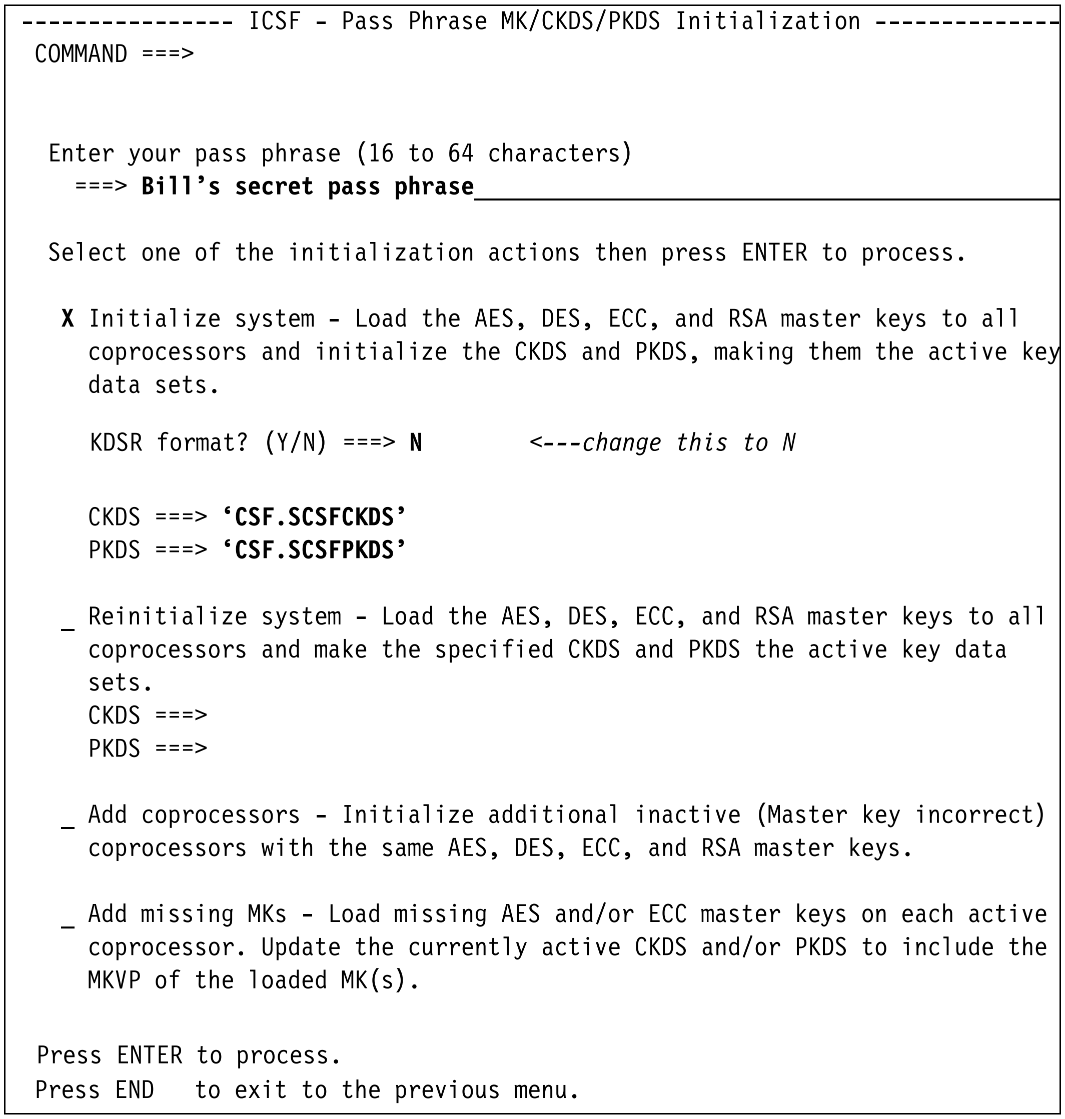
Figure 17-2 Verifying that the cryptographic co-processor is online

3. Press F3 to return to the first ICSF panel and select option 6. With this option, you can enter a *passphrase* that is automatically used to initialize the basic co-processor master keys. (An alternative method for initializing master keys is to use multiple other functions on the ICSF panels. Unless you are familiar with cryptographic co-processor management, use the simple passphrase initialization process.)

4. Complete the passphrase panel, as shown in Figure 17-3 (using your own passphrase). You must enter your passphrase and two data set names; specify the `no KDSR` format; and enter a character to select the `Initialize system` option. The two data set names that shown in the example (`CSF.SCSFCKDS` and `CSF.SCSFPKDS`) are preallocated but empty in the current ADCD z/OS system. (This situation might change in newer z/OS ADCD releases. Prototype jobs to create these data sets can be found in `SYS1.SAMPLIB` members `CSFCKDS` and `CSFPKDS`.) These data set names should match whatever you specified in `PARMLIB`.

   Note the passphrase that you selected, including uppercase and lowercase, spaces that you entered (including at the end of the phrase), and any special characters that are involved. If you have encrypted data involving the adapter functions and want to reinitialize the adapter controls, re-enter the *same* passphrase.

   The sample jobs to allocate these data sets use the names `CSF.CSFCKDS` and `CSF.CSFPKDS` while other jobs use `CSF.SCSFCKDS` and `CSF.SCSFPKDS`. Note the additional "S" in these names. We changed the sample allocation jobs to use the "S" versions of the names. There is no requirement to use particular names if you are consistent.



```
---------------- ICSF - Pass Phrase MK/CKDS/PKDS Initialization -------------
 COMMAND ===>


  Enter your pass phrase (16 to 64 characters)
    ===> Bill's secret pass phrase____________________________________________

  Select one of the initialization actions then press ENTER to process.

   X Initialize system - Load the AES, DES, ECC, and RSA master keys to all
     coprocessors and initialize the CKDS and PKDS, making them the active key
     data sets.

     KDSR format? (Y/N) ===> N        <---change this to N


     CKDS ===> 'CSF.SCSFCKDS'
     PKDS ===> 'CSF.SCSFPKDS'

   _ Reinitialize system - Load the AES, DES, ECC, and RSA master keys to all
     coprocessors and make the specified CKDS and PKDS the active key data
     sets.
     CKDS ===>
     PKDS ===>

   _ Add coprocessors - Initialize additional inactive (Master key incorrect)
     coprocessors with the same AES, DES, ECC, and RSA master keys.

   _ Add missing MKs - Load missing AES and/or ECC master keys on each active
     coprocessor. Update the currently active CKDS and/or PKDS to include the
     MKVP of the loaded MK(s).


 Press ENTER to process.
 Press END   to exit to the previous menu.
```

Figure 17-3 Passphrase panel

5. Start ICSF again and select option 1. Enter the letter `S` in front of the `7C00` co-processor name and press Enter. This action should produce a display showing some of the master key initialization.

The basic cryptographic co-processor setup is complete. Do not experiment with option `2` (`MASTER KEY MGMT`) functions unless you are certain that you know what you are doing.

Tip: The instructions that provided here are intended for customers with little or no experience with the IBM zSystems cryptographic adapters. Experienced customers might handle the emulated cryptographic adapter initialization differently.

17.4 Operational notes

CSF must be started (as a started task) to use it. At the time of writing, this task is automatically done in the latest ADCD z/OS releases. You can easily verify the CSF operation by going to **`omvs`** and issuing the following command:

```
od -An -N4 -td /dev/random
```

If CSF is not working, the result is an internal error message. If these functions are working, a random number is displayed.

The ISV zPDT cryptographic co-processor emulation functions are intended for use by developers who require these functions. It should be clearly understood that while using ISV zPDT that these emulated functions are not intended to produce a secure system or function as a secure peer when dealing with private data.

17.4.1 Multiple ISV zPDT instances

ISV zPDT can have multiple instances in operation. These instances can share facilities, such as shared direct access storage device. If shared facilities are used, then an ISV zPDT controller instance[6] must be present, as described in Chapter 10, “Multiple instances and guests” on page 215. Co-processors that are defined in the controller instance can be shared by all ISV zPDT instances. A maximum of 16 co-processors may be present in a “normal” ISV zPDT instance. A maximum of 64 co-processors may be defined for a controller instance. (Any ISV zPDT configuration that involves so many crypto co-processors is unusual.)

An additional devmap statement is used in the devmap of an ISV zPDT instance that is using co-processors that are defined in the controller instance:

```
domain <member name> a y   #(“a” is a coprocessor number, and y is a domain
number.)
```

The member name is required if the domain statement appears in the controller instance. The name is used if the domain statement is in an operational instance devmap. The domain number (`y` in the statement above) can be a single number, a list of numbers that are separated by commas, or a range of numbers that are separated by a dash. The angle brackets around the member name are not part of the syntax. They indicate an optional parameter. The domain numbers must be specified in the ICSF startup parameters and are different for each z/OS instance.

[6] A controller instance is an ISV zPDT instance (with a separate devmap and started with an **`awsstart`** command) that does not contain any IBM zSystems processors.

Three shared cryptographic co-processors, which are used by three ISV zPDT instances, might be defined as follows:

```
#------------- controller instance ----------------------
[system]              #No processor is defined for the controller.
...
...
[adjunct-processors]
crypto 0
crypto 1
crypto 2

#------------ ISV zPDT instance 1 ---------------------------
[system]
processors 1
...
[adjunct-processors]
domain 0 1
domain 1 1
domain 2 1

#------------ ISV zPDT instance 2 ---------------------------
[system]
processors 1
...
[adjunct-processors]
domain 0 2                    #All the domain statements
domain 1 2                    #could be in the controller
domain 2 2                    #devmap instead. Your choice.

#------------ ISV zPDT instance 3 ---------------------------
[system]
processors 1
...
[adjunct-processors]
domain 0 3                    #If the domain statements are in the
domain 1 3                    #controller devmap, they need the relevant
domain 2 3                    #member name as the first parameter.
```

Each ISV zPDT instance has a different domain number that is specified in the domain statements. In this example, the domain numbers are the same as the instance numbers, but this situation is a coincidence. ISV zPDT supports a maximum of 16 domains for each emulated co-processor.

17.4.2 Co-processor control commands

A number of commands are included for specialized management of the ISV zPDT cryptographic co-processors. These commands are issued from a Linux terminal window. ISV zPDT must be operational for these commands to be used. They are *not* needed for normal system use, and as a best practice, do not experiment with them unless you have a good understanding of what you are doing. In the following commands, the *n* variable is the cryptographic co-processor number and the *y* variable is a domain number.

Briefly, the commands are as follows:

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

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

- This command queries basic status and domain information. With no operand, it lists all the co-processors that are available. With an operand, it lists which domains are used by the indicated co-processor.

  ```
  $ ap_query
  $ ap_query -a n
  ```

- This command creates an (emulated) cryptographic co-processor:

  ```
  $ ap_create -a n
  ```

- This command removes the indicated co-processor process if it is not connected to a CP process:

  ```
  $ ap_destroy -a n
  ```

- These commands vary online or vary offline connections between co-processors and their processing queues. The optional **y** operand specifies a domain number.

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

- This command lists Vital Product Data (VPD) for the indicated co-processor:

  ```
  $ ap_vpd -a n
  ```

When an ISV zPDT instance is started (while processing the devmap), an **ap_create** command is automatically issued for that instance. If this instance is a stand-alone ISV zPDT instance, **ap_von** commands are issued for all domains. (It is not issued for a controller instance.) If this instance is an ISV zPDT instance that uses shared co-processor resources, **ap_von** commands are issued for the co-processors and domains that are specified in the devmap.

The "real" cryptographic co-processors on large IBM zSystems machines have similar control functions, but they are performed in different ways. Do not attempt to use these commands as listed here on larger machines.

17.4.3 New z/OS releases

The co-processor master keys, which are stored in the Linux `srdis` subdirectory, must be consistent with the data in the `CSF.SCSFCKDS` and `CSF.SCSFPKDS` data sets in z/OS.[7] If you install a new z/OS release and create z/OS data sets, you must initialize the new data sets or copy the contents of the old data sets to the new data sets. Unfortunately, these data sets are Virtual Storage Access Method (VSAM) data sets, which makes moving them or copying them more complicated.

[7] You might have used different data set names.

If you plan to work with encrypted data (as opposed to developing programs that use encryption functions), you must carefully plan backups for the co-processor data (in the `srdis` subdirectory) and the z/OS data sets that are used by CSF. The ISV zPDT functions have no special way to recover lost encryption keys.

17.4.4 Programming with CSF

Here are the core elements of a trivial program that uses the cryptographic co-processor (through a CSF programming interface) to obtain random numbers:

```
*
* GET RANDOM NUMBERS AND PRINT THEM
*
         LA    7,20                GET 20 RANDOM NUMBERS
LOOP1    CALL CSNBRNG,(RETC,REASC,EXDL,EXD,FORM,RANNUM)
         CLC   RETC(4),SZEROS
         BNE   ERROR2
         LA    1,RANNUM            WHERE TO START HEX CONVERSION
         BAL   10,AHEXLINE
         MVC   PRINTLNE(80),SBLANKS
         MVC   PRINTLNE(21),=C'RANDOM NUMBER (HEX) ='
         MVC   PRINTLNE+22(16),SWOUT
         PUT   PRINTD,PRINTLNE
         BCT   7,LOOP1
         ...
* VARIOUS WORK AREAS AND CONSTANTS
PRINTLNE DC    CL80' '
RETC     DC    F'0'                RETURN CODE (ICSF)
REASC    DC    F'0'                REASON CODE (ICSF)
EXDL     DC    F'0'                EXIT DATA LENGTH (ICSF)
EXD      DC    CL4' '              EXIT DATA (ICSF)
FORM     DC    CL8'RANDOM  '       RULE FORM
RANNUM   DC    2F'0'               RANDON NUMBER
         ...
//L.SYSLIB DD DISP=SHR,DSN=SYS1.MACLIB
//         DD DISP=SHR,DSN=CSF.SCSFMOD0
//G.SYSPRINT DD SYSOUT=*
```

17.5 Basic data set encryption example

Attention: The examples that are shown in this section were created under an older z/OS system than what is current when ISV zPDT GA11 was released. You might need to make minor changes in the following outlines.

Many ISV zPDT customers also use the ADCD z/OS package. The following instructions provide two options for basic implementations of data set encryption.[8] One option uses RACF and a specific High-Level Qualifier (HLQ) to select data sets for encryption, and the other option uses JCL parameters to control encryption. The setup is essentially the same for both options.

[8] Later z/OS ADCD releases might include some of the RACF work that is described in this section.

These examples ignore all the planning and controls that are used in a production environment and concentrates only on a basic setup for data set encryption. There are many ways to accomplish the various steps that are needed, and the following instructions outline one of these ways. Some details might be different for later ADCD releases.

This example assumes a basic systems programming familiarity with z/OS and the z/OS ADCD systems. The following names are used in the example:

- PROT is an arbitrary choice, and it is used both as a HLQ and as a Storage Class name, but there is no need to use the same name for both in your own work.
- PROTECT is used as a Storage Group name.
- SYS1.SOW1.SCDS and SYS1.SMS.CNTL exist in the current z/OS ADCD.
- DB2STORG and DB2STORC are member names that are used in the current z/OS ADCD.
- ADCDA and ADCDB are user IDs (in the current ADCD) with no privileges.
- The address (AB0) that is used for a new volume is arbitrary within ADCD input/output definition file (IODF) constraints.
- The new volume volser (SMS001) is arbitrary.
- MYPROT.KEY1 is an arbitrary label for an encryption key.
- PROT.TRY1 is an arbitrary data set name with HLQ PROT.
- OGDEN.TRY2 is an arbitrary data set name without a special HLQ.
- The Linux directory /z/zos23 represents the working directory for running ISV zPDT, but users might have their ISV zPDT startup in any directory.

One goal in our example is to use RACF controls to automatically encrypt any data set with the HLQ of PROT. The other goal is to specify encryption by using additional JCL parameters. In both cases, an encryption key with the label MYPROT.KEY1 is used. IBMUSER is the administrator (with RACF SPECIAL authority). User ID ADCDA is allowed access to the key and the data sets. User ID ADCDB is allowed access to the data sets but not the key.

Volume setup

Complete the following steps:

1. You need a DFSMS-managed volume for extended data sets. Create this volume before starting ISV zPDT with the following command:

   ```
   $ alcckd /z/zos23/SMS001 -d3390-1   (choice of a 3390-1 is arbitrary)
   ```

2. Add this volume to your ISV zPDT devmap, and also add a crypto adapter to the devmap:

   ```
   [system]
   memory 8G
   3270port 3270
   processors 3

   [adjunct-processors]
   crypto 0
   ...
   [manager]
   name awsckd 0001
   ...
   device AB0 3390 3990 /z/zos24/SMS001 # (The AB0 address is not special.)
   ...
   ```

3. After starting z/OS, initialize the new volume as SMS-managed:

```
//INITVOL  JOB  1,OGDEN,MSGCLASS=X,REGION=40M
//  EXEC PGM=ICKDSF
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  INIT UNIT(AB0) NOVALIDATE NVFY VOLID(SMS001) PURGE -
  VTOC(0,1,14) STORAGEGROUP
/*
```

4. Vary the new volume online.

SMS setup

You must make the new SMS-managed volume usable, which involves defining a Storage Group (containing the new volume) and a Storage Class for the target data sets. These controls are the minimum SMS controls that are required. While working with ISMF, complete the following steps:

1. Start ISMF. If you cannot see option 6, select option `0 (ISMP Profile)`, then option `0 (User Mode)`, and then option `2 (Storage Administrator)`. Exit from ISMF and start it again.
2. Create a Storage Group that is named PROTECT. Select option `6 (Storage Group)` and enter `'SYS1.SOW1.SCDS'` as the CDS Name. Enter `PROTECT` as the Storage Group Name and `POOL` as the type, and then select option `3 (Define)`. Change any of the characteristics that you want. In our example, we changed only Guaranteed Backup Frequency to `NOLIMIT` because ISMF needed an entry for this parameter. Select option `5 (Volume)` and then option `2 (Define)`. Enter your selected volser (`SMS001` in this example) in a prefix line.
3. Create a Storage Class named `PROT`. Select option `5 (Storage Class)` and enter `'SYS1.SOW1.SCDS'` as the CDS Name. Enter `PROT` as the Storage Class Name, and select option `3 (Define)`. We changed the Guaranteed Space parameter to `Y`.
4. Update the Storage Class and Storage Group ACS routines.[9] Select option `7 (Automatic Class Selection)`. Enter `'SYS1.SOW1.SCDS'` as the CDS Name and select option `1 (Edit)`. This action opens the ISPF edit panel. Enter the name `'SYS1.SMS.CNTL(DB2STORG)'` as the data set name and press Enter. Add the following lines at the beginning of the `WHEN` section:

```
WHEN (&HLQ = 'PROT')
   DO
      SET &STORGRP = 'PROTECT'
      EXIT
   END
WHEN (&STORCLAS = 'PROT')
   DO
      SET &STORGRP = 'PROTECT'
      EXIT
   END
```

5. Press PF3 and then edit `'SYS1.SMS.CNTL(DB2STORC)'` by adding the following lines:

```
WHEN (&HLQ = 'PROT')
   DO
      SET &STORCLAS = 'PROT'
      EXIT
   END
```

[9] The ACS routines that are distributed with new z/OS ADCD releases tend to vary, and the examples in this publication might need to be adapted to new ADCD releases.

6. Go to the end of the DB2STORC member and look for an **OTHERWISE** statement:

```
      OTHERWISE
         DO
            SET &STORCLAS = &STORCLAS      (it was &STORCLAS = '')
            EXIT
         END
```

7. Exit the edit panel and select 2 (Translate) in the ACS Application panel. Set the SCDS name to 'SYS1.S0W1.SCDS', the ACS Source to 'SYS1.SMS.CNTL', and Member to DB2STORG. Place any name for the Listing Data Set, and press Enter. If the results are good, repeat the process for member DB2STORC. (The statements after **OTHERWISE** in DB2STORC are odd, but they are needed to accept a storage class that is entered in a JCL **DD** statement.)
8. Update the Control Data Set. Select ISMF option 8 (Control Data Set). The CDS name is 'SYS1.S0W1.SCDS'. Select option 5 (Activate). Place a '/' character as indicated and press Enter.
9. Exit ISMF.

RACF work

While working as IBMUSER, complete the following steps for RACF:

1. Using ISPF option 6, verify that classes CSFKEYS and CSFSERV are active, generic, and RACLISTed:

```
SETROPTS LIST          (Look through the output to verify the details.)
```

2. Check access to RACF class CSFSERV by using ISPF option 6:

```
RLIST CSFSERV *
```

On some older systems, we found that the access was UACC(READ), which is not a good idea. If the UACC needs changing, try the following commands:

```
RALTER CSFSERV * UACC(NONE)
SETROPTS RACLIST(CSFSERV) REFRESH
```

3. User IDs ADCDA and ADCDB might need their passwords reset. To do so, use the following commands:

```
ALU adcda PASS(xxxxxx) NOEXPIRED     (Use the appropriate passwords.)
ALU adcdb PASS(yyyyyy) NOEXPIRED
```

4. Data set encryption requires the user to have read access to the following profile in the FACILITY class:

```
STGADMIN.SMS.ALLOW.DATASET.ENCRYPT
```

For the z/OS 2.3 ADCD, we found that this class was set for IBMUSER. You can verify this setting by running the following command:

```
RLIST FACILITY STGADMIN.*
```

You need to grant **PERMIT** access to users (other than IBMUSER) that need to access data set encryption for this profile.

5. Working as IBMUSER, select ISPF option 6 and define a number of RACF profiles:

```
RDEF CSFSERV CSFKGUP  UACC(NONE)    needed for KGUP
RDEF CSFSERV CSFKGN   UACC(NONE)    needed for KGUP
RDEF CSFSERV CSFKRR2  UACC(NONE)    needed for KGUP
RDEF CSFSERV CSFKRC2  UACC(NONE)    needed for KGUP
RDEF CSFSERV CSFPMCI  UACC(NONE)    needed to initialize ICSF
RDEF CSFSERV CSFOWS   UACC(NONE)    needed to initialize ICSF
RDEF CSFSERV CSFDKCS  UACC(NONE)    needed to initialize ICSF
RDEF CSFSERV CSFCRC   UACC(NONE)    needed to refresh ISCF
SETROPTS RACLIST(CSFSERV) REFRESH
```

As the profile owner, `IBMUSER` has **`ALTER`** access to these profiles.

6. While you are working with RACF, use the following job to enable automatic encryption for the HLQ that you selected (PROT) and allow access to the key that you intend to use:

```
ADDGROUP PROT        (Add a group so that the HLQ can be protected.)
ADDSD 'PROT.*' UACC(NONE) DFP(DATAKEY(MYPROT.KEY1))
PE 'PROT.*' ID(ADCDA) ACCESS(READ)
PE 'PROT.*' ID(ADCDB) ACCESS(READ)
RDEF CSFKEYS MYPROT.KEY1 UACC(NONE) ICSF(SYMCPACFWRAP(YES) -
       SYMCPACFRET(YES))
PE MYPROT.KEY1 CLASS(CSFKEYS) ID(ADCDA) ACCESS(READ)
SETROPTS RACLIST(CSFKEYS) REFRESH
```

The **`PERMIT`** statements allow ADCDA and ADCDB access to `PROT.*` Data sets are not needed because there are no RACF profiles to prohibit their access. However, this action illustrates what would be done in a more general environment. `IBMUSER`, as the profile owner, has access to these profiles.

ICSF and master keys

Initialize ICSF and the master keys as described in 17.3, “Initial ICSF startup” on page 333 if you have not done so.

Creating an encryption key

Create a random secure key by using a batch job, and then list the key data set with an IDCAMS job:

```
//CRYPTO2 JOB  1,OGDEN,MSGCLASS=X
//  EXEC PGM=CSFKGUP
//CSFCKDS  DD DISP=OLD,DSN=CSF.SCSFCKDS
//CSFDIAG  DD SYSOUT=*,LRECL=133   (The LRECL parameters are required.)
//CSFKEYS  DD SYSOUT=*,LRECL=1044
//CSFSTMNT DD SYSOUT=*,LRECL=80
//CSFIN    DD *
   ADD TYPE(DATA) ALGORITHM(AES) LABEL(MYPROT.KEY1) LENGTH(32)
/*
//*
//  EXEC PGM=IDCAMS
//IN1  DD  DISP=SHR,DSN=CSF.SCSFCKDS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
   PRINT INFILE(IN1) DUMP
/*
```

Refresh the in-storage CKDS by going to ICSF and selecting option 8 and then option 4, which is `REFRESH`. Enter `'CSF.SCSFCKDS'` (with single quotation marks) as the New CKDS parameter and press Enter. (The title is misleading because `CSF.SCSFCKDS` is not new in this case.) Exit ICSF.

Testing and verifying encryption

Test the RACF-controlled encryption with a job like the following one:

```
//CRYPTO3 JOB 1,OGDEN,MSGCLASS=X
//  EXEC PGM=IEBGENER
//SYSPRINT DD *
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.ASM        (Select a simple sequential file.)
//SYSUT2 DD DISP=(NEW,CATLG,DELETE),UNIT=3390,DSN=PROT.TRY1,
//         SPACE=(TRK,(10,1)),VOL=SER=SMS001,DSNTYPE=EXTREQ
```

Submit the job. If it runs without errors, go to the new `PROT.TRY1` data set. You should be able to go to it because it is automatically decrypted for `IBMUSER` because `IBMUSER` has access to both the data set and the key.

Log in as `ADCDA` and go to the data set. This action should work because ADCDA has access to both the data set and the key. Log in as ADCDB and go to the data set. This action fails because ADCDB does not have access to the key.

To determine whether the data set is encrypted, as `IBMUSER`, run the following job:

```
//CRYPTO4 JOB 1,OGDEN,MSGCLASS=X
//  EXEC PGM=ADRDSSU,REGION=0M
//SYSPRINT DD SYSOUT=*
//DDNAME DD DISP=SHR,DSN=PROT.TRY1
//SYSIN DD *
   PRINT INDDNAME(DDNAME) DS(PROT.TRY1)
/*
```

The results should show the data set as it is stored (encrypted) on disk.

Instead of using RACF to control encryption (through a preselected set of HLQs), you can specify encryption by using only JCL control. In this case, we do not use the HLQ of `PROT` in our example:

```
//CRYPTO5 JOB 1,OGDEN,MSGCLASS=X
//  EXEC PGM=IEBGENER
//SYSPRINT DD *
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.ASM       (select a simple sequential file)
//SYSUT2 DD DISP=(NEW,CATLG,DELETE),UNIT=3390,DSN=OGDEN.TRY2,
//         SPACE=(TRK,(10,1)),VOL=SER=SMS001,DSNTYPE=EXTREQ,
//         STORCLAS='PROT',DSKEYLBL='MYPROT.KEY1'
```

You can again verify the encryption by dumping the data set with **`ADRDSSU`**. You might also use TSO commands such as the following ones to verify that the catalog entry denotes encryption and the key label:

```
LISTC ENTRIES('OGDEN.TRY2') ALL
LISTC LEVEL(PROT) ALL
```

Try logging on to TSO as user `ADCDA` and go to the two data sets (`PROT.TRY1` and `OGDEN.TRY2` in the examples). You should be able to access them. Then, log in as `ADCDB` and try it. You cannot access them because `ADCDB` does not have access to the crypto key.

Re-initializing the master keys

While experimenting with your sandbox ISV zPDT system, you might want to re-initialize th master keys. You might find this task difficult because the CKDS and PKDS data sets have content in them and a master key exists. To correct this situation, complete the following steps:

1. Stop CSF with a **`P CSF`** command.
2. Use an ISV zPDT command (entered from Linux) to clear the master keys:

   ```
   $ ap_zeroize -a 0 -d 0              (Assume crypto adapter 0 and domain 0.)
   ```
3. You might need access to a specific RACF profile. If so, use the following commands:

   ```
   RDEF CSFSERV CSFRSWS UACC(NONE)      (As the owner, you have ALTER access.)
   SETROPTS RACLIST(CSFSERV) REFRESH
   ```
4. Using ISPF 3.4, delete the VSAM clusters `CSF.SCSFCKDS` and `CSF.SCSFPKDS`. (Using a **`DEL`** line command instead of a **`D`** line command is a best practice.)
5. Allocate new data sets with a job like the following example. (Prototype jobs can be found in `SYS1.SAMPLIB` members `CSFCKDS` and `CSFPKDS`).

   ```
   //CSFDS  JOB  1,OGDEN,MSGCLASS=X
   //DEFINE EXEC PGM=IDCAMS,REGION=0M
   //SYSPRINT DD SYSOUT=*
   //SYSIN DD *
    DEFINE CLUSTER(NAME(CSF.SCSFCKDS) VOLUME(B5SYS1) -
        RECORDS(200 100) RECORDSIZE(272 2048) KEYS(72 0) -
        FREESPACE(10 10) SHAREOPTIONS(2,3)) -
     DATA (NAME(CSF.SCSFCKDS.DATA) BUFFERSPACE(100000) ERASE) -
     INDEX (NAME(CSF.SCSFCKDS.INDEX))
   DEFINE CLUSTER(NAME(CSF.SCSFPKDS) VOLUME(B5SYS1) -
        RECORDS(200 100) RECORDSIZE(800 3800) KEYS(72 0) -
        FREESPACE(0 0) SHAREOPTIONS(2,3)) -
     DATA (NAME(CSF.SCSFPKDS.DATA) BUFFERSPACE(100000) ERASE CISZ(8192)) -
     INDEX (NAME(CSF.SCSFPKDS.INDEX))
   /*
   ```
6. Start CSF (**`S CSF`**) and continue with the master keys.

17.5.1 z/VM usage

Cryptographic co-processors are defined for z/VM guests as shown in the following example:

```
USER USERJOE 999999 512M 16E BDEG
  ACCOUNT ABC123 ABC123
  CRYPTO DOMAIN 13 14 15          (domains to be used)
  CRYPTO APDED 1  3               (co-processors to be dedicated for use)
  OPTION TODENABLE MAINTCCW
  MACHINE ESA 64
  CPU 0 BASE
  CPU 1
  IPL 190 PARM AUTOCR
etc
```

Server Time Protocol

Server Time Protocol (STP) provides synchronized time-of-day (TOD) clocks among several connected systems. IBM Z Program Development Tool (IBM zPDT) GA6 (and later) provides this function for use among multiple zPDT systems. The zPDT implementation is not designed for connection to larger zSystems STP configurations. Figure 18-1 provides an overview of the function.

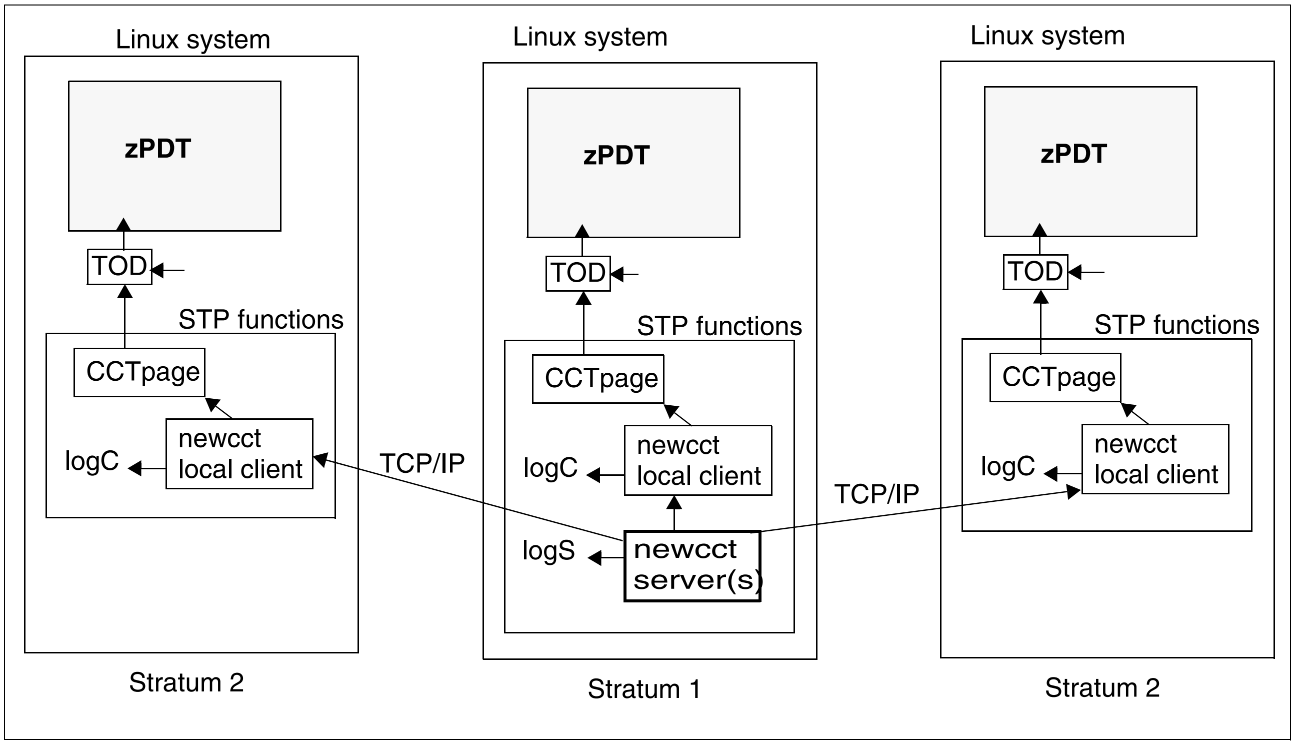


Figure 18-1 STP overview

An STP implementation is more complex than indicated in Figure 18-1, but it is useful as a starting point in describing the function.

Two terms are important in this description:

- Coordinated Timing Network (CTN): Indicates a collection of servers that are time-synchronized. On larger zSystems machines, the coordination can be provided by STP or (on older systems) by a Sysplex Timer or both. For zPDT, only STP is involved.
- Server Time Protocol (STP): Describes the protocol operating in and between the servers in the CTN. In a rough sense, STP is an implementation of a CTN.

STP and CTN terminology is sometimes used interchangeably, which is technically incorrect but the meanings are usually clear. Figure 18-1 on page 347 shows three zPDT instances (each in a separate PC) in a CTN. Figure 18-1 on page 347 includes the following high-level elements:

- A CTN has stratum levels. Stratum level *n* provides timer control to level *n+1*. Stratum 1 is the top layer in a CTN environment and must exist. In practice, a zPDT CTN must have a stratum 1 server and one or more stratum 2 clients. Stratum 3 clients (fed by stratum 2 clients) are possible but are not described here and were not tested.
- The stratum 1 server might obtain its TOD from various sources, including the machine BIOS clock, Network Time Protocol (NTP) connections, or an external high-precision clock. These sources can feed or initialize the Linux clocks, and then STP synchronizes with the Linux clocks.
- The STP server (in the stratum 1 machine) connects to clients. These client are *local clients* (in the same Linux as the server) or *remote clients* (in other machines that are connected by TCP/IP).
- STP servers and clients provide log functions. Our implementation (described later) places them in `/tmp` by default, but this location can be changed.
- The STP executable program is named **`newcct`**. The same program is used for the server and clients.
- An STP client maintains a small file that is named `CCTpage`, which is in `/tmp` by default. It contains local timer adjustment data, and it is updated as needed by the STP client.
- Information from `CCTpage` is used to adjust the TOD clock that is used by zPDT. Multiple elements contribute to the TOD clock that is used by the STP server, including these elements:
  - The Linux REALTIME clock (maintained by Linux software)
  - The Linux MONOTONIC clock (maintained by Linux software)
  - The PC BIOS hardware clock, which is referenced when Linux starts
  - The PC time-stamp counter (TSC), if present
  - The `libCCT` library package and the `ReadCCT` routine

  The interaction of these elements is complex and not described in this document.
- Figure 18-1 on page 347 shows a single zPDT instance in each machine. Multiple instances are possible, and each instance requires a separate STP client instance.
- The STP functions run at the Linux level. The functions are started before staring zPDT. In practice, the STP functions are typically started when Linux starts and are left running while Linux is running.
- The remote clients use TCP/IP connections to the server. In a larger zSystems environment, these connections are typically through Coupling Facility (CF) channels, which are high-speed connections. The accuracy and usability of the zPDT STP implementation might be affected by delays in the IP network.
- The zPDT STP implementation cannot be connected to a larger zSystems CTN.

This chapter describes the practical configuration and operation of STP in a zPDT system. For more information about in general, see *Server Time Protocol Planning Guide*, SG24-7280.

18.1 CCT uses

The coordinated TOD that is provided to multiple z/OS systems can be used for various purposes, such as later comparisons or reconciliation of log files. However, the most common use is for sysplex operation. A z/OS sysplex, either a basic sysplex or an IBM Parallel Sysplex, must have coordinated TOD clocks. For zPDT, Parallel Sysplex operation is provided only when all z/OS systems are guests within a single z/VM environment. In this case, a simulated clock is used, and a CCT is not relevant.

18.2 Configuration

To use the STP function, complete the following steps:

1. Create and customize the `/usr/z1090/bin/CCT_data` file while working as root. The pattern is in `/usr/z1090/bin/CCT_data.MASTER`, which contains the following text:

   ```
   ThisNode = xxx.xxx.xxx.xxx
   MasterNode = yyy.yyy.yyy.yyy
   CCTid = zPDTbasic
   ClientLogEnabled = Y
   CCTdir = /tmp
   ```

   – `ThisNode` is the address of your Linux system. It may be expressed as an IP address or as a domain name. For all parameters, a space must exist before and after the equal sign.

   – `MasterNode` is the address (IP address or domain name) of the Linux system running the STP server. `ThisNode` and `MasterNode` are the same value for the Linux system running the server, and it also is a client function.

   – `CCTid` is the name of your STP network. This parameter should be the same for all your systems. The sample name, `zPDTbasic`, is an arbitrary but workable name.

   – `ClientLogEnabled` must be set to `Y` or `N` to control client logs. If enabled, a line is added to each client log every minute. These logs might be useful when setting up a basic sysplex complex, but might not be useful in an established operational system.

   – `CCTdir` indicates where the log files and an internal file (`CCTpage`) file are placed. A good place is `/tmp` unless you have specific requirements.

   – The log files (if enabled) have names indicating the starting date and time of that file. Existing log files are overwritten.

   – Place a space before and after the equal sign in this file. The IP addresses are for the base Linux systems, and not for the z/OS systems.

   In our example, we copied `/usr/z1090/bin/CCT_data.MASTER` to `/usr/z1090/bin/CCT_data` and working as *root*, configured the following lines for our first Linux server:

   ```
   ThisNode = 192.168.1.80        (Our second Linux server uses 192.168.1.90.)
   MasterNode = 192.168.1.80
   CCTid = zPDTbasic
   ClientLogEnabled = Y
   CCTdir = /tmp
   ```

2. Use the **stpserverstart** command to start the script. The command also automatically adds commands to the Linux **cron** function so that the server is started automatically when Linux starts or fails. The **stpserverstop** command can be used to stop the server and remove the automatic **cron** function. In our example, we start our first Linux server because it contains the STP server. The **stpserverquery** command may be used to determine the state of your CCT/STP network. After the command is added to the **cron** function, the STP clients and server are automatically started whenever you start Linux. It is not harmful if you are not using the command for zPDT.
3. Update your devmaps to include an `[stp]` stanza. After this stanza is included, zPDT cannot be started with this devmap unless the CCT function is operational. You might want to keep a separate copy of your devmaps without the `[stp]` stanzas for use in a stand-alone mode.

   ```
   [stp]
   ctn 00000000F1F0F9F0       #16 hex digits beginning with 00
   node 1 W520 *              #asterisks marks this node
   node 2 W510
   ```

   There is one node statement for each Linux PC in your complex. The number (1 or 2) indicates the stratum level of the machine. You should have one stratum 1 server and one or more stratum 2 clients. The names of the nodes (W520 and W510 in the example) are arbitrary, but z/OS messages might be more meaningful if these names are the names of the Linux systems, as expressed in `/etc/HOSTNAME`.

   The asterisk (*) denotes the current system that is processing this devmap. In the example, our second system has the asterisk after the W510 name. Except for the location of the asterisk, all the zPDTs in your complex have the same `[stp]` stanza in their devmap.
4. The z/OS **CLOCKxx** members that you use (in PARMLIB) must be altered to use the STP function:

   ```
   (Member CLOCKB1 in USER.PARMLIB in our test system)
   OPERATOR NOPROMPT
   TIMEZONE W.05.00.00
   ETRMODE NO
   ETRZONE NO
   ETRDELTA 10
   STPMODE YES
   STPZONE NO
   ```

   STP operation in z/OS can be verified with the MVS command **D ETR**:

   ```
   D ETR
   .....
   SYNCHRONIZATION MODE = STP
   .....
   ```

You cannot use a devmap (as the operand of an **awsstart** command) that has an `[stp]` stanza unless the **CCT/STP** function is running. You can check the status by using the **stpserverquery** command.

If you display Linux processes directly with, for example, a **ps aux | grep cct** command, you see a line that ends with `newcct` information, as in the following example:

```
newcct 192.168.1.80 -L -e1000 -E300000 -f -I zPDTbasic -n0 -r120
newcct -v1
```

The CCT/STP function does not affect the Linux TOD clock. It affects only clocks that use the CCT interface library routines; in practice, the zPDT (when a `[stp]` stanza is present in the devmap) and z/OS (when the **`CLOCKxx`** parameter is set).

18.3 More details

The z/Architecture places tight tolerances on STP clock synchronization and uses high-speed coupling channels as part of the design to achieve the specified tolerances. zPDT, by using IP communication, cannot duplicate this environment, so the z/Architecture tolerances are relaxed for the zPDT STP function. You must evaluate the usefulness and correctness of the zPDT STP function for your environment.

Both the client and server create log files. Our sample configuration places them in `/tmp`. Existing log files are overwritten when a server or client is started. The server log files typically have little in them. A client log file might have entries such as the following ones:

```
Wed Oct  8 11:14:54 EDT 2014        (<-- this is when this log file started)
%CCT (TAI seconds)      Offset (us) Dispersion   Skew (ppm) Steering   RTT (us)
 1412781391.767832          -0.088       0.009        0.00      0.00     0.228
 1412781451.774840          -0.102       0.006        0.00      0.00     0.183
 1412781511.782228          -0.103       0.006        0.00      0.00     0.181
 1412781571.789614          -0.097       0.026       -0.00      0.00     0.212
```

With the default parameters, a client log line is written every minute. As a general statement, you do not need any information from the logs except an indication of how often the client might have failed. (The default parameters cause the client to restart itself after a failure.) The log files are named `STPServerLog` and `STPClientLog` and placed in the Linux directory that you specify in the STP configuration file. These two files are open whenever the STP function is active. If you want to capture the contents of a log file for later inspection (they are overwritten whenever the STP function is started), use the Linux **`cp`** command, as in this example (the **`cp`** command may be used against an active Linux file):

```
$ cp /tmp/STPClientLog /tmp/STPClint.monday3PM
```

The fields in the client log are as follows:

International Atomic Time (TAI)	Number of seconds since 1 January 1972, including leap seconds. (The leap second data is not normally available to the zPDT **`newcct`** program, and this definition might not be completely accurate.)
Offset	The difference (in microseconds) between the master CCT clock (the stratum 1 server) and the client clock, as calculated by the client.
Dispersion	The uncertainty in the calculated offset value.
Skew	The difference in the clock stepping rate between the master and the subordinate.
Steering	The rate at which the client clock is being steered to reduce the offset toward zero.
RTT	The minimum measured round-trip delay for a set of CCT sample “ping-pong messages”. It is the sum of the minimum forward delay time and minimum backward delay time for a set of CCT sample ping-pong messages,

Note: "Ping-pong" messages are short message exchanges between a client and server.

The TOD that is provided to z/OS by the `CCT/STP` function ultimately depends on the TOD that is returned by the effective Linux clock.[1] A user might have a substantially inaccurate TOD in the PC BIOS clock that is used to set the Linux clocks when Linux starts. Another user might have an NTP[2] connection for Linux, resulting in a Linux clock that is accurate to within a second or so. At the zPDT level, there is no control over the accuracy of the time data that is provided by Linux. The STP function keeps multiple zPDT TOD values synchronized, but does not control the accuracy of the time that is presented.

The log files are not intended for normal user inspection and may contain messages with whimsical phrasing, such as this example:

```
Incomplete PingPong exchange on client: Resource temporarily unavailable
```

This message indicates a client failure, but the client immediately restarts itself. We have seen these messages at random intervals on zPDT CCT networks that are closely coupled (private 1 GB Ethernet), but we have not seen them on other systems that are connected through the internet. In our test environment, these transient failures appear to have no effect on the z/OS basic sysplex operation. Corresponding log messages on the CCT server include the following ones:

```
Failed to read doorbell msg
Welcome                        (Denotes a client restart.)
```

When a remote client connects to the STP server, another server instance is automatically started. No action is needed to manage this task, but you might see multiple STP servers when using the Linux **ps** command.

If you have an `[stp]` stanza in a devmap, you cannot start this devmap (by using the **awsstart** command) unless you have an STP function active. If you do not have STP active, the zPDT startup fails. The most common error message is `INVALID REGISTER NUMBER`.

The zPDT STP facility relies on a fast, low-jitter TCP/IP connection between host Linux systems to synchronize the zPDT TOD clock values. Insufficient TCP/IP connection quality results in STP reporting either unsynchronized or unusable TOD clock-source machine checks to the zPDT-hosted OS. You must provide a physical TCP/IP connection with sufficient quality to support machine-check-free STP operation. In a virtualized environment, the hypervisor increases the TCP/IP latency and jitter, as perceived by the STP facility, which increases the risk of disruptive zPDT STP machine checks. We have performed limited STP testing in a virtualized environment with mixed results, so as a best practice, do not use STP in a virtualized environment.

[1] This topic is complex because Linux has several clocks. We ignore this detail here.
[2] NTP references an accurate TOD service that is available over the internet.

18.3.1 Leap seconds

Tip: There is no need to deal with leap second details unless you (or your applications) are sensitive to them. We expect 99% of zPDT users will ignore leap seconds.

You may update the leap second information block by including an extra statement in the devmap, as in the following example:

```
[stp]
ctn 00000000F1F0F9F0       #16 hex digits beginning with 00
node 1 W520 *              #asterisks marks this node
node 2 W510
LEAPSECONDS 25 26 2015 6 30 23 59 59
```

The format for the **LEAPSECONDS** statement is as follows:

```
LEAPSECONDS <active> <new> <year> <month> <day> <hour> <minute> <second>
```

- `<active>` is the number of leap seconds present.
- `<new>` is the replacement number of leap seconds. It is not an extra value, but a complete replacement of the active number. It potentially can be a smaller number than the `<active>` value, although this situation has not happened yet. The difference between `<active>` and `<new>` should never be more than 1.
- `<year><month><day><hour><minute><second>` is the time at which the `<new>` value is effective.

Two new zPDT commands (issued in a Linux command-line interface (CLI)) are provided for leap second details:

```
$ d lso                     (displays the leap second information block)
$ st lso <active><new><year><month><day><hour><minute><second>
$ st lso 25 26 2015 6 30 23 59 59     (an example)
```

We understand that the z/OS **CLOCKxx** parameter **STPZONE** must be set to `YES` for z/OS to process leap seconds when using an STP time source. The leap second information block is displayed, as in the following example:

```
LEAP SECOND OFFSET INFORMATION BLOCK
word(0):             80000000
lso active:                 25
lso new:                    26
update time:   73061E173B3B00  - 2015:  6:30:23:59:59
```

19

Problem handling

This chapter uses the term *ISV zPDT service provider*. IBM is not a formal “service provider” for Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT), but an equivalent service might be provided (as an option) by an ISV zPDT vendor. Not all ISV zPDT users might have such service providers, and some of the information in this chapter might not apply in these cases. Also, various web online forums or discussion groups might informally help with problems.

The underlying Linux systems and the IBM zSystems operating system and applications that are used, are not a formal part of ISV zPDT. Your “service provider” options might include assistance with expanded software issues.

Various chapters throughout this publication contain text that might help with different problem areas. This chapter summarizes higher-level steps for dealing with some common issues.

19.1 Problems starting an ISV zPDT operation

Many problems are related to device map (devmap) errors, which are often due to errors in Linux file names in the devmap. The solution is to check the devmap carefully, remembering that file names are case-sensitive in Linux. Use the **awsckmap** command for some basic devmap checking.

Problems with obtaining a license (from the ISV zPDT token) are typically due to an expired or unactivated token.

Messages about time cheat errors indicate a more serious problem. These errors are typically caused either by moving a token between multiple PCs that do not have their time-of-day (TOD) clocks closely synchronized, or manipulation of the TOD clock on the personal computer (PC).

The ISV zPDT system uses several Linux shared storage (virtual memory) areas. These areas are normally freed when ISV zPDT ends with an **awsstop** command. A failure or incorrect handling during ISV zPDT operation might result in these shared storage areas not being released, which can prevent ISV zPDT from being started again. There are Linux commands to individually free shared storage areas, but they require multiple detailed steps. Restarting Linux is a primitive but effective way to solve this situation.

If you have problems with the token or license, complete the following steps before seeking help:

1. Run the **uimcheck** command.
2. Verify that the local token can be accessed by using the following Linux commands:

   ```
   $ lsusb | grep Rain                         (You see "Rainbow Technologies".)
   $ ps -ef | grep usbdaemon | grep -v grep    (You see safenet_sentinel.)
   $ ps -ef | grep Sntl | grep -v grep          (You see SntlKeysSrvrlnx.)
   # netstat -tlp                        (Misc, including remote license servers)
   ```

3. Start ISV zPDT with a simple devmap:

   ```
   [system]
   memory 4096m
   processors 1
   3270port 3270
   ```

4. If a license problem persists, delete `/usr/z1090/uim/uimclient.db`. You need root authority to do this task. Then, restart ISV zPDT.
5. Run following command. Save the output if further help is needed.

   ```
   $ cat  /usr/z1090/bin/sntlconfig.xml
   ```

If you seek help, provide the following information:

- Has your ISV zPDT system worked before the current problem? If so, what changed?
- What are the software levels that are involved?
  - What is the exact ISV zPDT level? The level is a number like `z1090-1-11.57.03.x86_64`. To determine the ISV zPDT level, use the following Linux command:

    ```
    $ rpm -qa | grep 109          (Use "109", not "1090" or "1091".)
    ```

  - What is the exact Linux distribution name and level?

  – What is the IBM zSystems operating system level (and is it an Application Development Controlled Distribution (ADCD) system)?

- Are you running a virtual or container environment?
- How much memory is available on the PC (or your virtual machine (VM) or container)?
- What type of token do you have (a 1090 or 1091)? Is it initialized? Are you certain? (Use your ISV zPDT provider to initialize 1090 tokens. If it is a 1091 (indicating an IBM Z Development and Test Environment (ZD&T) system), then contact your IBM ZD&T vendor.
- Are you using a remote license server (as opposed to a token on your PC)?
- Are you trying to use a cloned system for the first time?

19.2 Problems during an ISV zPDT operation

The ISV zPDT system maintains several logs and traces during operation. The ISV zPDT programs might detect a problem and capture the logs or traces at the time of the problem.

Attention: The zPDT logs and traces are meaningful only to the zPDT developers. They are not as useful for normal zPDT operation.

Over time, there can be a buildup of unwanted files in the logs directory. Assuming that you are not working on an outstanding ISV zPDT problem, you can delete all the files in the logs directory (do this task when ISV zPDT is not running). An easy way to do this task is to use the **`--clean`** option of the **`awsstart`** command. Again, assuming that you are not working with an existing ISV zPDT problem, you might use the **`--clean`** option every time that you perform an **`awsstart`**. Conversely, do not use the **`--clean`** option while you are working on a problem because some of the older log and trace files might be wanted later.

If device managers fail, they automatically create a trace file and are automatically restarted. This restart will not occur more than three times per minute. If a fourth failure occurs within the same minute, the device manager is not restarted, and the devices that it controls become *not operational* for the remainder of that ISV zPDT session. The ISV zPDT design limits the size of the logs and traces and should never create more than about 30 MB per emulated device (and there is normally much less than this amount).

19.3 Core images

Severe problems might cause Linux *core image* files to be created. If they are created by ISV zPDT, they should go into the log subdirectory and be cleaned up with the **`--clean`** option of **`awsstart`**. If you are actively working on a problem with your ISV zPDT provider, these files might be useful. Otherwise, they can be deleted because they can be rather large and might create a disk space problem.

Consistent dumps (core images) when ISV zPDT is started can occur if you have a many emulated I/O devices (more than 100, for example) and you have not made memory management adjustments.

If you cannot start ISV zPDT (with the **`awsstart`** command), there might be a core image file. In this case, the core image file might help with problem analysis, and it should be preserved while the problem is under investigation.

19.4 Logs

The term “log” can have multiple meanings in a Linux environment. Among these meanings are the classic Linux logs (in `/var/log`) and the specific ISV zPDT logs (in `/home/ibmsys1/z1090/logs`, if user ID `ibmsys1` is used for ISV zPDT operation). The logs that might be helpful for initially looking at a problem are the following ones:

```
/var/log/message                                (for Linux messages)
/home/ibmsys1/z1090/logs/log_console_......txt   (for ISV zPDT messages)
```

The `log_console` names tend to be long because they include a process ID (PID) and time and date, as in this example:

```
log_console_p6780_t2014-11-21_13-31-07.txt
```

19.5 Emulated volume problems

An emulated 3390 volume is a single Linux file that was created with the **`alcckd`** command. Three variations of this command are useful for problem handling:

```
$ alcckd /z/WORK03 -rs           (Scans the emulated volume for format errors.)
$ alcckd /z/WORK03 -rf           (Replaces a bad track with zeros.)
$ alcckd /z/WORK03 -r            (Displays a volser and its size.)
```

The **`-rs`** function scans the emulated volume and verifies that it is in the correct 1090 emulation format.[1] The **`-rf`** function replaces improperly formatted tracks with a properly formatted track containing zeros. The original contents of the track are lost, but the remaining function of the volume is maintained.

Assuming that your emulated 3390 volumes are in the `/z` directory and there are no other file types in this directory, you can verify the format of all the volumes with the following Linux shell commands:

```
$ cd /z                         (location of emulated CKD, and nothing else)
$ for i in *; do alcckd $i -rs; done
```

The **`ckdPrint`** command can be used to examine the contents of an emulated CKD volume. This command prompts for the beginning cylinder and head numbers and the ending cylinder and head numbers. These numbers are in decimal format. The following example lists the records on cylinder 0, head 0 of volume Z9SYS1:

```
$ ckdPrint /z/Z9SYS1
DeviceType 3390, Cylinders-3339, Trks/Cyl-15, TrkSize-56832
Input extent in decimal - CC-low HH-low CC-high HH-high
00 00 00 00
....
....
```

The **`tapeCheck`** command can be used to verify the format of an awstape file. This command reads the awstape file and verifies that the awstape control blocks are logically correct.

[1] Only the emulation format is checked. There is no check for data content or operating system metadata (label, Volume Table of Contents (VTOC), and such).

19.6 Special problem-related commands

A number of ISV zPDT commands are intended to be used only at the direction of IBM development personnel, and they supply the specific commands and operands to be used. This category includes the following commands:

```
$ awslog                    (including --logsize and --logcount operands)
$ doOSAcmd                  (various subcommands)
$ dreg                      (shared resource registry)
$ dshrmem                   (shared memory)
$ printlog                  (only for some .gz logs)
$ printtrace                (only for some .gz traces)
$ qpci                      (Displays zEDC information.)
```

The contents and formats of the various log and trace files are not documented and intended only for diagnostic use. Our experience is that these files are not useful for solving general user-level problems.

19.7 Linux monitoring

Some monitoring of Linux statistics might be helpful. The **vmstat** command is useful and causes little interference with other processes in Linux. Here is its syntax:

```
$ vmstat -w 3 4                        (4 samples at intervals of 3 seconds)
procs -----------memory------------ ---swap--- -----io----- --system-- ----cpu----
 r b    swpd   free   buff    cache   si     so    bi      bo    in     cs us sy id wa st
 2 0      0 397600 101936 2398160     0     0   109     52  154   350 14  2 81  3 0

r = number of Linux processes waiting for run time
b = number of Linux processes sleeping
swpd = amount of swap space used (KB)
free = amount of idle memory (KB)
buff = amount of memory used as buffers (KB)
cache = amount of memory used as cache (KB)
si, so = Linux paging activity in KB/second
bi, bo = I/O activity in blocks/second
in = interrupts per second
cs = context switches per second
us = percent of CPU time in user mode
sy = percent of CPU time in kernel mode
id = percent of CPU time idle
wa = percent of CPU time waiting for I/O
st = timeslices stolen for virtual machine
-w = creates a wider report line with better column organization
```

Important data from **vmstat** includes the Linux swap rates (`si, so`). Any number here can degrade ISV zPDT performance. If ISV zPDT suffers a page fault, then the whole IBM zSystems CP operation must wait for the page fault to be resolved. The `wa` statistic (CPU percentage waiting for I/O) provides an indirect indication of I/O overload.

The **top** command provides data about individual Linux processes. We are most interested in the `emily` process (which is the name of the Linux process representing the CP), but information about various device manager processes can be useful in spotting bottlenecks. Enter **q** from the keyboard to terminate the **top** command:

```
$ top
top - 11:14:14 up 1:24,   3 users,  load avg: 1.26, 1.15, 1.02
tasks: 128 total, 1 running, 121 sleeping, 6 stopped, 0 zombies
CPU(s): 49.9% us, 0.2% sy, 0.0% ni, 49.7% id, 0.0% wa 0.0% hi, 0.0% si
mem:  3111660k total, 2719204k used,  392456k free,  105356k buffers
swap: 2104504k total,       0k used, 2104504k free, 2397740k cached

 PID  USER    PR   NI   Virt   RES   SHR   S  %CPU  %MEM   TIME    COMMAND
7040  ibmsys1 25    0  1549m  1.5g  1.5g   S   100  49.6  35:41.2  emily
7051  ibmsys1 16    0  1510m  36m    36m   S     0   1.2   0:03.1  awsckd
```

19.8 Summary

The material that is presented in this chapter (and in this publication) is intended to point to some basic elements that might help resolve ISV zPDT issues or provide starting points for investigating problems. Before requesting help from a service provider (if one exists) or posting problems on various websites, make a sincere effort to reduce the description of a problem to the most basic components.

A

FAQ

The following frequently asked questions (FAQs) and discussions might be helpful to newer Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) users.

General

Q: Many of the zPDT modules and commands are named aws****. What does “aws” mean?
A: It stands for *A*dvanced *W*ork*S*tation. It is an IBM convention that such things as program module names start with an identified prefix, and “aws” was chosen for zPDT.

Q: You say “ISV zPDT” almost everywhere in this publication. Can any of the materials be used for IBM Z Development and Test Environment (ZD&T)?
A: Most of it can, but you might require specific IBM ZD&T information and documentation. The two primary zPDT products are not always synchronized for changes, and they can differ in various details. This publication is mostly specific to the development level of the ISV zPDT version.

Q: For some years, I saved the z/OS Application Development Controlled Distribution (ADCD) files (in gzip format) on DVDs. However, I seem to have problems now. Why?
A: DVD formats, as generally used on Linux systems, cannot handle files larger than about 4.2 GB. More recent z/OS ADCD systems sometimes have compressed files larger than this amount, which creates errors or failures when attempting to “burn” the file on a DVD. USB flash drives, in their typical format when purchased, can have the same problem. It is possible to change the format of a USB flash drive (to ext4, for example) to avoid this problem.

Q: Is this system a multi-user one?
A: Yes. Multiple Time Sharing Option (TSO) users can connect in several ways and use the system in a normal manner. The same applies to z/VM users, CICS users, and others.

Q: How many users can the system support?
A: There is no definitive answer. The aws3274 device manager supports 32 connections (it emulates local 3270 devices). There is no specific maximum for TCP/IP (awsosa) connections to z/OS. Practical performance is the primary limitation, not the theoretical connectivity for terminal connections. A system might do well for one heavy IBM Db2 or Java user, 10 - 20 typical TSO users, or 100 web users with a low transaction rate to a z/OS web server. The answer to the question depends on the nature of the workloads that are involved.

Q: On my Linux system, I log in as root and then **`su`** to another user ID when necessary. For example, I use **`su ibmsys1`** when using ISV zPDT. Is there a problem with this process?
A: As a best practice, do *not* do this action. Operating as root when it is not necessary is a security risk. Also, you must understand the difference between **`su ibmsys1`** and **`su - ibmsys1`**.[1] The first version (without the “-” character) retains root’s environment while working as `ibmsys1`. The second version (with the “-” character) switches to the `ibmsys1` environment, which is the version that should be used if you really must log in as root.

Q: When using a remote license server, are there any security exposures?
A: This topic is complicated. As a security measure, starting a fix for ISV zPDT GA10 remote license server access is done with HTTPS instead of HTTP.

Q: IBM zSystems Integrated Information Processor (zIIP) processors are now “free”, but IBM zSeries Application Assist Processor (zAAP) processors are not. Why?
A: The usage of zAAPs is deprecated and does not apply to IBM z14 or later.

Q: Will adding zIIPs make my system faster if I have sufficient cores?
A: Maybe, depending on the workload. However, with some z/OS parameter combinations (in workload manager (WLM)), if zIIPs exist, some workloads are forced to run *only* on zIIPs, even if there are idle CPs. (Recent z/OS ADCD releases have such parameters.) This topic is a complex z/OS topic that is not unique to ISV zPDT.

[1] There is nothing special about user ID `ibmsys1`. It is the user ID that we use in all ISV zPDT examples, but it is not reserved or special in any way.

Q: Chapter 8, “ISV IBM Z Program Development Tool licenses” on page 179 is a bit confusing. Are hardware tokens being replaced with “software” licenses?
A: No. At the time of writing, the “software-only” licenses are only for IBM ZD&T. In general, the software-only license servers are more complex to install and manage, and might have additional fees that are associated with them.

Q: Can I install (with `gunzip`) another z/OS volume while ISV zPDT is running?
A: In principle, yes. In practice, it is not a good idea. Disk usage during the compression of a volume is intense and can generate Missing Interrupt Handler (MIH) messages from z/OS.

Q: Does the number of ISV zPDT licenses that are available (assuming a remote license server) equal the number of IBM zSystems serial numbers that are assigned?
A: No. For more information, see 8.4, “General ISV zPDT client and server details” on page 184.

Q: I purchased an extra ISV zPDT license to configure a zIIP. What has changed in ISV zPDT?
A: A zIIP no longer needs a separate ISV zPDT license. You might use your extra license to configure an extra CP.

Q: Can I install (decrypt) the z/OS ADCD residence volume from the relevant IBM site (intended for users with 1090 tokens) while using a 1091 token (or software license equivalent)?
A: No. The appropriate token (1090 or 1091) must be used with the appropriate z/OS ADCD (or other authorized) distribution.

Q: Can a z/OS program easily detect whether ISV zPDT or IBM ZD&T is being used?
A: In both cases, the emulated IBM zSystems CPs specify model type 1090. In a practical sense, one might determine which is being used by considering direct access storage device volume serial numbers, parmlib contents, and others, but this technique assumes an understanding of the practical layout of the two packages.

Q: Are new IBM zSystems processor instructions (as provided with recent new IBM zSystems machines) present?
A: Yes. The emulated instruction set matches the architectural level state for a release of ISV zPDT. A few instructions dealing with functions that are not present in an ISV zPDT environment are not available. In rare cases, ISV zPDT might defer implementation of an especially complex instruction to later releases of ISV zPDT, but this situation should not inhibit “normal” usage of z/OS.

Q: In one place, you say that more cores than CPs are required, but in another place, you say that an equal number is required. Which is correct?
A: The number of cores *must* be at least equal to the number of CP + zIIP + IBM Integrated Facility for Linux (IFL) + zAAP processors. Use more cores than CPs+zIIPs+IFLs+zAAPs to provide better performance.

Q: Do I need to change any z/OS parameters to operate with ISV zPDT?
A: In principle, this situation is unlikely. In practice, you might need to adjust a few parameters that are primarily related to performance. For example, the CICS transaction timeout value might need to be increased for “heavy” transactions. Obviously, adjustments might be needed for widely different configurations. For example, moving a large Db2 application from a 1000 GB logical partition (LPAR) to a 16 GB ISV zPDT laptop might lead to difficulties.

Q: I have ISV zPDT and a hardware key. Where can I download z/OS?
A: z/OS (or any other IBM zSystems software) is not part of the core ISV zPDT program. For more information, contact your ISV zPDT provider.

Q: Do I need the hardware token to *install* ISV zPDT?
A: No, you need it only to *run* ISV zPDT or *install* ADCD z/OS initial program load (IPL) volumes.

Q: Can I use ICKDSF with the **ANALYZE** function for emulated CKD volumes?
A: No, in most cases. Emulated CKD devices (such as 3390) do not contain spare cylinders and diagnostic cylinders that might be required for the **ANALYZE** operation. The ISV zPDT command **alcckd -rs /z/B3RES1**, for example, performs a slightly similar function to verify that the basic 3390 track formats are correct on a volume.

Q: How accurate are the IBM zSystems time-of-day (TOD) and timer functions?
A: To a large extent, they are approximately as accurate as the timer in the underlying personal computer (PC). Some interval measurements might have a granularity of about 500 microseconds (plus the IBM zSystems operating system time that is needed to manage time-related activities). Do not depend on fine timing measurements on ISV zPDT to reflect what timing might be on a larger IBM zSystems system. Also, when a heavily loaded ISV zPDT system runs for a long time, there might be a slight loss of synchronization between the base PC TOD and the computed ISV zPDT TOD.

Q: I need to have multiple levels (more than three) of z/OS available for testing, although each z/OS is usually idle at any time. A 1090-L03 seems to be too much for my modest processing needs and 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 solution requires some z/VM skills. The solution requires more IBM zSystems memory than other potential solutions to avoid excessive z/VM and z/OS paging. (For more information about older z/OS releases, see 3.2, “System stanza” on page 50.) Memory is important for reasonable performance in such situations. However, you must be sensitive to the architectural level of the ISV zPDT system that you are using. For example, the IBM z14 level (ISV zPDT GA8 and GA9) cannot be used with older operating systems that do not support this level. Also, “overloading” your ISV zPDT environment can create performance and reliability issues.

Q: Why do you not provide a definite million instructions per second (MIPS) or MSU value? These values would help me determine how to best use ISV zPDT.
A: There are no definite numbers. A MIPS measurement depends on the exact workload and your system configuration. A workload description like “I have six TSO users” is useless.

Q: Why do some ADCD releases pause for many seconds while shutting down?
A: You can edit the `SHUTDOWN` entries in `PARMLIB` to remove or change any pause statements. Some functions, such as zFS, have built-in delays that we cannot change.

Q: You are inconsistent with the addresses for the ADCD volumes. For example, sometimes volume SARES1 is at address A91 and sometimes at address AA0. Which is correct?
A: Both are correct. Any 3390 volume can be at any address that is defined as a 3390 in the input/output definition file (IODF) for that z/OS system. For ease of documentation, we always show the primary IPL volume at A80 and the SYS1 volume (which contains the IODF and IPLPARM data sets) at A82, but these addresses are not required. The IPL address and parameter must match the addresses that you use. We tend to show SARES1, if mounted, at various addresses.

Q: What happens if I remove the hardware key?
A: The ISV zPDT functions stop after a while.

Q: You use user ID `ibmsys1` throughout all the examples. Is there something special about this user ID?
A: No. However, you should always use the same Linux user ID when running ISV zPDT, and the user ID should not be longer than 8 characters. (If a **system_name** statement is used in the device map (devmap), then there is no special limit on the Linux user ID name.)

Q: Can I use an alternative translation table to convert Extended Binary Coded Decimal Interchange Code (EBCDIC) to ASCII for awsprt output?
A: No. However, the translation table was updated for ISV zPDT GA9.

Q: My z1090 rpm installation failed with an error message about `db_recovery`. What now?
A: Run the command **`rpm --rebuilddb`** and then install ISV zPDT 1090.

Q: I am using an emulated printer that sends output to a Linux file. Does this file remain open for output by ISV zPDT while it is running?
A: Yes. However, it is closed if you use **`awsmount`** to assign a new output file for the printer.

Q: Are ISV zPDT commands case-sensitive? Can I issue **`ipl`** or **`IPL`**?
A: The command names are case-sensitive. They are the names of Linux files, and Linux file names are case-sensitive.

Q: The **`z1090instcheck`** command does not work. Why?
A: You might need the full path name for the **`z1090instcheck`** command if your Linux **`PATH`** environmental variable does not include `/usr/z1090/bin`.

Q: The ADCD z/OS system always starts TCP/IP and associated jobs. How can I delete them?
A: You can edit the VTAMAPPL types of entries in `PARMLIB` and remove the associated start commands. While running z/OS, you can issue **`P TCPIP`**. You might need to cancel address spaces that are related to TCP/IP, such as `INETD1`.

Q: Can I use IBM Resource Measurement Facility (RMF)?
A: Yes, but not all of RMF is relevant on an ISV zPDT system, and some functions might be misleading.

Q: How do I perform an IPL of the SARES1 volume?
A: Assume that it is mounted at A9C. The command is **`ipl A9C parm 0A9CSA`**. Use **`S SHUTSA`** to shut down the SARES1 system. (The system can be mounted at any address that defined as a 3390 in the ADCD IODF.)

Q: Why is the first part of the z/OS IPL sequence a little slow under z/VM? After this portion is complete, z/OS seems to run at a more normal speed under z/VM.
A: This situation is due to memory initialization and management functions being initialized through multiple virtualization paths (virtual machine (VM), **`START INTERPRETIVE EXECUTION`** (SIE), and Linux). The time seems related to the defined z/OS guest memory size and disk cache performance. The effect is less apparent when larger PC memory is available or solid-state drives (SSDs) are used.

Q: I want to place Db2 buffers in the Coupling Facilities (CFs) in my Parallel Sysplex. Also, I want to put the Job Entry Subsystem (JES) 2 checkpoint data and RACF data there. How do I do this task?
A: These tasks are beyond the scope of this publication. Study the appropriate manuals and seek help from a systems programmer with experience in this area. You might also need to change the CF definitions memory sizes in the z/VM directory.

Q: I crash my ISV zPDT system (possibly with the **`awsstop`** command). Can I continue to take this action with the Parallel Sysplex running?
A: You can do it, but starting the Parallel Sysplex again might be difficult. Follow the normal shutdown procedures for a Parallel Sysplex. In particular, use the **`V XCF,xxx,OFFLINE`** command to stop more z/OS members of the sysplex.

Q: Why is the z/OS TOD clock (or the z/VM TOD clock) at least 27 seconds different than the base Linux clock?
A: IBM zSystems operating systems apply leap seconds to the clock, but Linux does not. The current leap seconds offset, at the time of writing, is 27 seconds. This IBM zSystems function is included with the ISV zPDT GA6 and later releases. Also, while ISV zPDT is running, additional time differences might accumulate.

Q: Can I emulate various subcapacity settings and controls?
A: No. ISV zPDT does not provide these functions.

Q: If I unplug my token, how long will ISV zPDT continue to operate?
A: The length of operation without a token (or remote license) is variable, but might be up to several minutes.

Q: I dislike Linux gnome. Can I use the Xfce desktop?
A: Yes, if you can find it for your Linux distribution. ISV zPDT is not sensitive to the Linux desktop that is used.

Q: This publication mentions various limitations for crypto usage. Will these limitations be resolved in future zPDT releases?
A: Possibly. However, it is unlikely that all the more advanced features of IBM zSystems system cryptoadapter functions will be emulated.

ISV zPDT configuration and devmaps

Q: Can I run z/VM, two z/OS guest machines, and a CP guest machine with only one PC processor ("core") and one ISV zPDT CP?
A: Yes, in theory. z/VM shares the processor with all the guest systems. However, this configuration can produce timeouts within z/OS. We consider it below the minimum level for practical use.

Q: Is a Parallel Sysplex system practical on a laptop?
A: Yes, but ensure that you have sufficient PC memory. Use 16 GB as a minimum in this case.

Q: I see that the z/OS ADCD system has IBM 3380 devices at some addresses. Do I need them?
A: These devices are present mostly for historical reasons, although it is possible that a few users need them for older software. Do not use them unless you must use IBM 3380 direct access storage device.

Q: Does IBM need to enable something to allow full operation of the IBM zSystems system internal crypto instructions?
A: No, full operation is always enabled.

Q: The basic ISV zPDT setup looks easy, although I realize the networking can be a challenge. However, reading Chapter 16, "Channel-to-channel" on page 323 (for simple JES2 networking, for example), Chapter 18, "Server Time Protocol" on page 347, or Chapter 17, "Cryptographic usage" on page 331 is confusing. Are these functions needed?
A: Good question. The cryptographic functions are used by a reasonable subset of ISV zPDT customers but are not "required" unless you have applications (programs or z/OS elements) that need them. The CTC information is used by few users, and the STP information by almost no one. They are included because they are part of the current IBM zSystems system basic design.

Q: Should device statements (in a devmap) be in order by addresses?
A: No particular order is required.

Q: I have volumes at addresses A80 - A8F. Do I need to define a new awsckd unit to add more disk volumes?
A: No, you can have up to 256 volumes in one instance of awsckd.

Q: All your examples have 3-digit emulated device addresses. Are they required?
A: No, you can use 3- or 4-digit addresses. The typical usage of only 3-digit addresses with the ADCD z/OS system is a historical accident.

Q: Can I use multiple ISV zPDT tokens to obtain more CPs?
A: Yes, up to a maximum of 8 CPs in an ISV zPDT instance. In this context, zIIPs, zAAPs, and IFLs count toward the limit of 8.

Q: Can ISV zPDT support older CKD drives, such as 3350?
A: No.

Q: Can I use MVS 3.8?
A: No. The ISV zPDT system does not support architectures before XA and 3380 and 3390.

Q: Can I configure ISV zPDT to act as, for example, a z800 system?
A: No. Each ISV zPDT release matches a specific IBM zSystems architecture. ISV zPDT GA8 matches IBM z14 systems, for example. There is no ISV zPDT facility to alter the architecture level.

Q: Is Flashcube supported for emulated disks?
A: No.

Q: Can emulated printer output be directed to `/dev/lp0` or similar?
A: We do not know because this scenario was not tested.

Q: What are the maximum numbers of CPs, ISV zPDT instances, and I/O devices?
A: A maximum of 8 CPs (or combinations of CPs, zIIPs, zAAPs, and IFLs) can be used in an ISV zPDT instance, although your license terms might have a lower limit. A maximum of 15 ISV zPDT instances can exist in a Linux system if you have sufficient licenses. A maximum of 2048 I/O devices can 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 to much smaller configurations.

Q: Can I move an ISV zPDT token between two PCs?
A: Technically yes,[2] but there is an important issue that is involved. The latest TOD value that is seen by the underlying PC hardware is stored in the token. If the token encounters an earlier time, it fails the operation with a time cheat message. If your two PCs have a significant time spread between their hardware TOD clocks, you might have problems.

Q: You have a whole chapter on STP and Coordinated Timing Network (CTN) (TOD functions) (Chapter 18, "Server Time Protocol" on page 347). Are these functions typically used with ISV zPDT?
A: As far as we know, they are rarely used with ISV zPDT. They are part of the IBM zSystems architecture and are included in ISV zPDT for this reason. Unless you have a specific requirement, do not experiment with these functions.

[2] You should observe the terms and conditions of your ISV zPDT license agreement.

Base Linux

Q: Why do you support only limited Linux releases?
A: IBM performs extensive testing for ISV zPDT. We use Linux releases that are current at the time that a release development starts. There are many practical reasons for not changing the Linux level midway in the development cycles.

Q: I have several Linux windows open while running ISV zPDT. I can enter ISV zPDT commands in any window, which is convenient. However, I also sometimes get output messages in a different window from where I entered a command. Is this normal?
A: Yes. ISV zPDT output messages (but not command output messages) are sent to the console session that issued the **`awsstart`** command.

Q: Can I make the **`kernel.shmmax`** value large to avoid worrying about it?
A: As far as ISV zPDT is concerned, you can. However, it is possible that other Linux applications might accept the large value and attempt to use unreasonable amounts of shared virtual memory, which results in excessive paging or system failure.

Q: Why is ISV zPDT placed in `/usr`? This directory should be only for basic Linux components.
A: You are correct, but it is difficult to change that situation.

Q: Can I run as root when installing and using ISV zPDT?
A: Yes, for part of the installation process. No, for operation. Follow the instructions concerning when to work as root and when to work under a normal user ID (such as `ibmsys1`).

Q: Does ISV zPDT operate in kernel mode? In suid mode?
A: A few administration modules operate in suid (root) mode. However, these details might change with future ISV zPDT releases. Starting with ISV zPDT GA10.2, there is no Linux root authority that is required during routine ISV zPDT operation.

Q: Can I routinely migrate to the next Linux releases when they become available?
A: Maybe. There is no unique ISV zPDT tie to a particular release, although the Linux release should be the same or later than the ISV zPDT “build” level. However, it is possible that the ISV zPDT installation steps might not work for a new release (due to different library paths or levels) or that the new release might not support the particular hardware in your base machine. Contact your ISV zPDT provider.

You are probably safe going to a new minor distribution release. For example, going from Red Hat Enterprise Linux (RHEL) 7.3 to RHEL 7.4 should be safe. It is a best practice to not jump to a new major distribution such as from RHEL 7.x to RHEL 8.x that is not listed as a “Base” level in 2.5, “ISV zPDT releases” on page 44. Our experience is that minor ISV zPDT installation adjustments are sometimes needed for new major Linux release levels.

Q: Is there any Linux checking that should be routinely done?
A: You should check your `/tmp` file system from time to time and ensure that free space is available for it. You might check your Linux home directory for ISV zPDT, which is `/home/ibmsys1` in our examples and delete any core files. The core files can be large and are unwanted unless you are actively debugging a problem. Core files in the ISV zPDT subdirectories should be investigated.

We have no specific recommendations about online updates to your Linux base system. In earlier years, we avoided these updates because of various problems that were introduced through them. More recently, some members of the ISV zPDT development team have been routinely doing online updates of their Linux (when ISV zPDT is not running) without experiencing problems that are related to the updates.

Q: Can I use compressed disk files or sparse files to reduce the amount of disk space that needed by z/OS volumes?
A: ISV zPDT does not support any form of compressed disks or files, or Linux sparse files. If you have a compressed disk scheme that is transparent to ISV zPDT, you can try it.

ISV zPDT operation

This section has frequently asked questions about the following topics:

- Tapes, SCSI drives, awstape, and disks
- PC hardware, cores, channels, and memory
- 3270 sessions and emulators
- Open Systems Adapter and LANs

Tapes, SCSI drives, awstape, and disks

SCSI tape drive support was dropped in ISV zPDT GA 10.1.

Q: I have a SCSI tape drive. I want to use it directly for Linux functions that are not connected with ISV zPDT operation, but I cannot find the **`mt`** command (a "standard" Linux command for manipulating tape devices).
A: We noticed that **`mt`** is not always installed with some Linux distributions. In some cases, it appears to be part of the **`cpio`** rpm.

Q: Can I use a SCSI digital linear tape (DLT) tape drive?
A: It should work if it supports the SSC-3 SCSI Command Set for Sequential Devices, although this scenario is not supported by IBM and has not been tested.

Q: Can I use a SCSI 4 mm tape drive?
A. It might work, but it is a best practice to not use 4 mm drives because they are poorly suited for emulated IBM S/390 work.

Q: Can awstape files from P/390 or R/390 systems be used with the ISV 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 that are created by ISV 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 that normally are used to indicate the logical end of data on a tape.

Q: How can I write a tape mark on an awstape volume?
A: Use **`awsmount`** ***`xxx`*** **`--wtm`**, where xxx is the address (device number) of the tape drive. There is a double dash before the **`wtm`** option.

Q: When performing an IPL of z/OS, I see messages beginning with **"**`CU AUTHORIZATION FAILED`**..."** for my 3390 emulated disk drives. What do these messages mean?
A: Ignore the messages. They occur because of emulation difficulties that do not affect the operation, and they will eventually be fixed.

PC hardware, cores, channels, and memory

Q: Most of the documentation is about large laptops or servers. I have a typical desktop PC. Can I use ISV zPDT with it?
A: Probably, assuming Linux works properly with the display, DVD drive, power, and local area network (LAN) interfaces on your desktop. You should have *at least* 8 GB of memory (more is better). However, the only formal support is for the machines that are described in this publication. IBM cannot undertake the extensive testing that would be needed to qualify the vast variety of PCs that exist.

Q: Can I use the Linux **taskset** command to bind a Linux core to an ISV zPDT CP? Would this setup make the hardware caches more efficient?
A: The ISV zPDT developers have not tried this action. The **taskset** command can bind a Linux process to a particular core, but it does not prevent other processes from being dispatched on that core, thus locking out the CP process that is bound to that core. Also, there are many ISV zPDT processes for I/O operation and Just In Time (JIT) functions. In a situation with sufficient cores that are available, you might improve ISV zPDT performance slightly, but this task probably would take considerable experimentation and be sensitive to a particular workload. Attempting to automate such tuning for all ISV zPDT installations and workloads can become complex and possibly degrade performance instead of enhancing it.

Q: Does ISV zPDT reserve PC processor cores for IBM zSystems emulated execution? Does it partition PC memory in some way to create IBM zSystems memory?
A: No. (See the prior question about the Linux **taskset** command.) A running ISV zPDT system consists of many processes and threads under Linux, which are dispatched in the normal Linux manner and reference Linux virtual memory as a normal application.

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 results in a slower system because the single processor must handle all IBM zSystems CP operations plus all the other processes for I/O and other Linux details.

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 because it might render your ISV zPDT token unusable. A powered USB port extender should work correctly.

Q: Can I use a USB disk drive for emulated IBM zSystems data?
A: Yes, assuming the base Linux recognizes and supports the drive in the normal manner. The drive might offer slightly less performance than the internal PC disk drives, but this situation might be acceptable in many cases.

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. ISV zPDT does not care about these settings because it uses Linux I/O functions.

Q: Does more PC memory improve performance?
A: Yes, up to a point. Linux can effectively use memory as a disk cache, which enhances performance.

Q: Is there an adapter for parallel channels?
A: At the time of writing, there are no hardware channel adapters for ISV zPDT systems.

Q: I have ESCON adapters from previous products for my Intel base PC. Can I use them?
A: No.

Q: Can a solid-state drive (SSD) be used (on the PC) instead of a traditional hard disk drive (HDD)?
A: Yes, they are effective.

Q: You frequently mention that paging should be avoided. Can I install more PC disks to reduce bottlenecks in this area?
A: We do not know. We doubt that another disk in a laptop or USB port would have much effect. More SCSI drives on a larger server, especially with multiple SCSI adapters, *might* help. We are interested in any documented experiences in this area.

Q: I am using the emulated crypto adapter, but RMF says that there is no adapter. What am I doing wrong?
A: RMF uses a **`CHSC`** command to obtain statistics from a crypto adapter. At the time of writing, ISV zPDT does not support this usage of the **`CHSC`** command. Use other means to verify that the crypto adapter is working as expected.

Q: I sometimes see a message beginning `AWSSTA138E TERMINATE CPU-0 FAILED, RC=43`. What does this mean? What should I do?
A: This message means that the zPDT license (typically from a token) was not obtained correctly or that it did not complete the initialization of the emulated CPU. You can check that your token license is valid; the token is correctly installed; and then try starting zPDT again.

3270 sessions and emulators

Q: How many "local" 3270 sessions can I use with ISV zPDT?
A: At the time of writing, the limit is 32 sessions with the aws3274 device manager. There is no limit on the number of sessions through z/OS TCP/IP.

Q: Can I use 3270 sessions on other PCs? Your example has all the sessions on the ISV zPDT machine.
A: Yes. Point your 3270 emulator (on your external PC) to the Linux IP address and port 3270 (if you are using port 3270 for aws3274). Use a relatively modern 3270 emulator, such as recent versions of IBM Personal Communications or x3270. Older, "free" 3270 emulators created problems during some of the ISV zPDT test cycles.

Q: Do the int3270port and intASCIIport interfaces provide the same functions as the equivalent functions on a Hardware Management Console (HMC) that is attached to a larger IBM zSystems system?
A: This operation is the intended one, although the operational characteristics might differ. These interfaces are normally used only when needed for installing a new IBM zSystems operating system. As a best practice, do not include them in a devmap unless there is a specific need for one of them, and do not include both in the same devmap.

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 differences in the handling of error and recovery situations.

Q: Does ISV zPDT handle 3270 nulls correctly?
A: This function is not one of ISV zPDT, but it is a function of the 3270 emulator and to some extent the application that is involved. Relevant functions for x3270 can be found by selecting **Options** → **Toggles** → **Blank Fill**. The IBM Interactive System Productivity Facility (ISPF) command **`nulls on|std|all|off`** might be relevant.

Q: I am using the IBM Personal Communications product to connect from a remote PC to z/OS running on ISV zPDT. Every time that I start Personal Communications, it wants to print something. How can I stop this action?
A: This issue is not related to ISV zPDT. Personal Communications stores user profiles in `.ws` files (such as `bill.ws`). Find the `.ws` profile that you are using and add the following lines at a reasonable place in the profile:

```
[LT]
IgnoreWCCStartPrint=Y
```

Q: I want to use Personal Communications instead of x3270. Is this approach acceptable? Can you include Personal Communications with the ISV zPDT package?
A: Personal Communications is part of a separate IBM product. We cannot include it as part of the ISV zPDT package.

Open Systems Adapter and LANs

Q: The **`--interface`** parameter for awsosa is confusing. How should I use it?
A: Read the material in Chapter 7, "Local area networks" on page 149. The **`--interface`** parameter was introduced because recent Linux distributions changed the way that LAN interfaces are named, so a more general method of specifying a LAN interface to awsosa might be needed.

Q: Are the `10.1.1.x` addresses for local 3270 sessions required? Can I use something like `192.168.123.5`?
A: The `10.x.x.x` addresses are not required. They are used in our examples to represent a simple setup. You should use non-routable addresses and make the devmap addresses (for the awsosa definition) match the z/OS addresses and match the 3270 emulator addresses.

Q: Connecting from a different PC (usually with a 3270 emulator) sometimes seems to produce security problems. Any suggestions?
A: There can be many parameters that can affect this situation, such as in z/OS, RACF, OVMS, Linux, routers, the web, and firewalls. The list is long and beyond the scope of the zPDT developers and this publication. However, you can look at some basic elements, such as an **`xhost +`** command on your remote connection machine (if that command even applies to your configuration).

Q: Will my devmap from a previous ISV zPDT release work with new ISV zPDT releases?
A: Probably. An important issue concerns path names for Open Systems Adapter (OSA) interfaces. Previous ISV zPDT releases (before GA10) considered only LAN interfaces that were not *down* when assigning path names. At the time of writing, the release considers all detected LAN interfaces, which can result in a different path name for OSA.

Q: Can I have multiple tap (tunnel) interfaces, such as tap0, tap1, and others?
A: Yes. A total of eight OSA devices of any type can be defined. The tunnel interfaces typically are CHIPD numbers A0, A1, A2, and so forth. The tap devices can be defined (as shown in the **`find_io`** command output), but they are not used unless a corresponding OSA device exists in the devmap. You can alter the CHPID numbers by using the **`--interface`** parameter on the awsosa device manager statement.

Q: Can I run multiple TCP/IP stacks on a single ISV zPDT emulated OSA-Express adapter?
A: Yes.

Q: Why might I need to specify a unit address in the device statements for OSA? I do not understand.
A: A full discussion is beyond the scope of this publication. As a best practice, use the QDIO (also known as OSA-Express Direct (OSD)) interface, in which case the unit addresses are not needed. For non-QDIO TCP/IP, ensure that the unit addresses are 0 and 1. (This configuration is required by the default OSA Address Table (OAT) that is used by OSA.) Ensure that unit address FE is used only for Open Systems Adapter/Support Facility (OSA/SF) when using the default OAT. The default unit address is the same as the low-order 2 digits of the device number ("address"). If you meet these requirements, there is no need to specify a unit address in the device statements for OSA. However, the older administration program for OSA/SF is not available in z/OS, and the replacement program is not usable with ISV zPDT.

Q: Does ISV zPDT support thin interrupts?
A: Yes, for OSA device emulation, but not for CF communication. (This function is known as the Adapter Interrupt Facility.)

Q: Can I filter IP traffic before it is sent to my emulated OSA-Express interface? This action reduces the overhead that is involved in rejecting packets that are not addressed to my system.
A: In OSD (QDIO) mode, there is some automatic filtering. In OSA-Express (OSE) (non-QDIO) mode, you can customize the OAT with your IP address. If you do this task, the OSA interface passes only packets that are intended for this IP address.[3] If this customization is not done (and it is not done in the default OAT), then all packets are sent to the host TCP/IP, and unwanted packets are rejected at that level. If you use Network Address Translation (NAT) functions on the base Linux, then most of the filtering is done at that level. At the time of writing, z/OS releases use an OAT management interface that is not supported by the zPDT.

Q: Is the OSN operation (CDLC) provided with current ISV zPDT OSA emulation?
A: No.

Q: Should I use 1492 or 1500 as the maximum transmission unit (MTU) size when using awsosa?
A: We use 1492. The details are beyond the scope of this publication. (As best we can tell, the IBM zSystems communication routines automatically adjust this number down if necessary, so it probably does not matter whether you specify 1492 or 1500.)

Q: Does current ISV zPDT QSA emulation include advanced functions such as VIPA?
A: Yes, when using QDIO.

Q: Can I use a continuing range of addresses (device numbers) when I have multiple OSA QDIO interfaces? For example, 400-402 for TCPIP1, 403-405 for TCPIP2, and so forth.
A: No. For z/OS, the first OSA address for a TCP/IP stack must be an even number. You would use 400-402, skip 403, then use 404-406, skip 407, and so forth. (This statement might not be correct for z/VM.)

Q: At the time of writing, do the ISV zPDT OSA offload functions work? Do they accomplish anything on an emulated system?
A: The Linux-based ISV zPDT OSA implementation does not use offload functions at the time of writing. In some cases, you might need to force Linux to disable offloading.

[3] However, the former program (OSA/SF) that was used to edit OATs is no longer available (starting with z/OS 2.3) and the replacement program (Queryinfo) is not supported by ISV zPDT.

Q: What PC card (PCMCIA-type card) should I use for additional Ethernet ports on a ThinkPad?
A. Use any card that the base Linux system accepts. Some years ago, we tried an Xterasys Gigabit PC Card (98-012084-585). We also informally tried several older IBM 10/100 EtherJet cards. However, none of them are "supported" ISV zPDT options, and we have doubts about more recent Linux support for such adapters.

Q: Can I use IP aliasing in Linux while using ISV zPDT?
A: Yes, but the alias addresses are not relevant to ISV zPDT and are not displayed by the **find_io** command.

Q: I have multiple Ethernet adapters, each on a different subnet. The response is slow and I get multiple responses to pings. Is there a problem with using multiple adapters?
A: In principle, no. However, multiple interfaces on different subnets should not be connected to the same virtual LAN (VLAN) because it creates routing, ARP, and duplicate response issues. Also, the external routing configuration (external to your system) might produce multiple responses. Multiple subnets on a single physical network might produce multiple responses. Do not use multiple LAN adapters unless you have the necessary networking skills.

In general, an ISV zPDT owner might require networking assistance for any configuration other than aws3274 emulation on a local router, which is not a unique zPDT issue.

Q: I have an error message about Generic VLAN Registration Protocol (GVRP) when I try to use a VLAN or VSWITCH in z/VM. Is this scenario supported?
A: No, GVRP is not supported. Specify **NOGVRP** for your VSWITCH. VLAN generally works, but there are exceptions.

Q: Is the OSA function for Integrated Console Controller (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 task while ISV zPDT is running? I am changing only Linux parameters, not OSA parameters.
A: This scenario is not supported or tested, and it probably will not work correctly. Do not change Linux LAN definitions while ISV zPDT 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. In principle, you also can use it for aws3274 clients if you find a client (and Linux host) that supports IPv6. As a practical matter, there has been minimal ISV zPDT testing of IPv6.

Q: You say that Ethernet Systems Network Architecture (SNA) operation is not *supported*. Might it work?
A: Yes, it *might*. It has not been tested and IBM does not respond to problems where it is used.

B

Non-QDIO Open Systems Adapter

The awsosa device manager can emulate QDIO or non-QDIO operation. The mode is selected by the **pathtype** parameter in the device map (devmap). OSA-Express Direct (OSD) specifies a QDIO operation, and OSA-Express (OSE) specifies a non-QDIO operation. A non-QDIO operation is also known as an *LAN Channel Station (LCS) operation* or *3172 operation*, although these descriptions are not exactly correct. A non-QDIO operation can involve TCP/IP, Systems Network Architecture (SNA), or both, although SNA usage is not tested or supported by Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT). Always use *OSD mode* unless you have a specific reason for using OSE mode. Multicast IP addresses are not supported for non-QDIO use.

IBM does not support Ethernet SNA operation for ISV zPDT, which means that problems or defects are not addressed by IBM if you attempt to use an SNA operation over Ethernet. However, some ISV zPDT users have worked with Ethernet SNA successfully. If you attempt this approach, consider the following information:

- The default Open Systems Adapter (OSA) Address Table (OAT) for the awsOSA device manager in LCS mode has TCP/IP at unit addresses 0 and 1. It has SNA only at unit address 2.
- Although SNA performance might be acceptable for simple testing, it is unlikely to be acceptable for heavier usage.
- A personal computer (PC) Ethernet adapter can 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, and the awsosa device manager is used in both cases.
- The Open Systems Adapter/Support Facility (OSA/SF) tool is no longer available with z/OS and the partial replacement (**QUERYINFO**) is not functional for ISV zPDT. The effect is that the default OSE (non-QDIO) configuration (the OATs) of an emulated OSA cannot be easily changed.
- As a best practice, use OSD (QDIO) unless you have a specific requirement for OSE mode, and can work around the **OSA/SF** and **QUERYINFO** limitations.

When using the non-QDIO interface to the emulated OSA-Express2 function, the key parameters might look like the following example:

```
Devmap

   [manager]
   name awsosa 22 --path=F0 --pathtype=OSE
   device E20 osa osa --unitadd=0
   device E21 osa osa --unitadd=1

z/OS TCP/IP Profile

   DEVICE LCS1 LCS E20 AUTORESTART
   LINK ETH1 ETHERNET 0 LCS1
   HOME 192.168.1.81 ETH1
   ...
   BEGINRoutes
   ;     Destination  Subnet Mask    FirstHop    Link Size
   ROUTE 192.168.1.0 255.255.255.0      =        ETH1 MTU 1492
   ROUTE DEFAULT                  192.168.1.1   ETH1 MTU DEFAULTSIZE
   ENDRoutes
   ...
   START LCS1
```

This example assumes that z/OS contains an appropriate channel-to-channel (CTC) or OSA definition for addresses E20 and E21.[1] Different addresses can be used, but they must match the IODF in your z/OS system. The `HOME` address and **`ROUTE`** statements in the example are only examples. The **`GATEWAY`** statements can be used instead of the **`ROUTE`** statements. The **`--unitadd`** parameter is used in the devmap because the default OSA unit addresses[2] would be 20 and 21 (by using the *two* low-order digits of the device number), and we want unit addresses 0 and 1.[3]

Which mode should you use for OSA connectivity? QDIO mode has many benefits for TCP/IP usage on a larger IBM zSystems server, for example, it reduces the IBM zSystems workload and provides automatic sharing of the adapter across multiple logical partitions (LPARs). These considerations do not fully apply to an ISV zPDT system. The following points are relevant:

- The QDIO operation offloads some processing from the ISV zPDT CP to the Linux processor. The offloading is not as much as on a larger machine, but it helps. The QDIO operation also reduces the number of IBM zSystems server instructions that are needed to maintain LAN I/O operation. In informal operation, we noticed that FTP performance was about 20% faster with QDIO than with LCS.
- QDIO operation is only for TCP/IP because it does not handle SNA. QDIO can provide VSWITCH, IPv6, and Enterprise Extension connections.
- Non-QDIO operation can mix TCP/IP and SNA if you want. However, SNA operation with ISV zPDT is not supported.
- Suitable non-QDIO (LCS) devices are defined in z/OS Application Development Controlled Distribution (ADCD) systems. (These systems are the CTCs starting at address E20.)

[1] Local area network (LAN) operation in LCS mode can use CTC definitions in the z/OS IODF. This function is a carryover from earlier LAN implementations.

[2] This unit address is the (emulated) hardware address within the (emulated) OSA control unit. It is not the device number (“address” in common terminology).

[3] The default OAT that is used by OSA requires unit addresses 0 and 1 for TCP/IP when in OSE mode.

Other than these points, there is no practical difference between using QDIO or non-QDIO on an ISV zPDT system. In particular, the user at a TN3270 Time Sharing Option (TSO) session cannot detect the difference. Normal TCP/IP functions, such as FTP and Telnet, do not detect any differences.

C

Generation 2 tokens and licenses

The license control mechanisms come from Gemalto N.V. (recently acquired by Thales) under the general product name of SafeNet.[1]

The IBM Z Program Development Tool (IBM zPDT) license tokens that are described in this publication (IBM part number 1090 for Independent Software Vendor (ISV) zPDT (ISV zPDT) and IBM part number 1091 for IBM Z Development and Test Environment (ZD&T)) belong to an older SafeNet generation of tokens. Informally, within this publication, we refer to them as Generation 1 ("Gen1") tokens. We were under the impression that the Gen1 tokens would be discontinued and replaced with a new set of tokens, which we informally named Generation 2 ("Gen2") tokens. More recent information indicates that the Gen1 tokens will be available longer than we anticipated, so there is little need for Gen2 information in this publication edition. Previous editions contained a number of references to Gen2 usage, which were removed (except for this small appendix) in the current edition.

If you are interested in Gen2 tokens, the following points might be useful:

- zPDT (both versions) will continue to support Gen1 tokens after the eventual production switch to Gen2 tokens. There is no need for existing zPDT users to obtain new tokens.
- Gen2 tokens have more options. At the time of writing, we expect that the same Gen2 token may be used for ISV zPDT or zD&T, depending on exactly which license material is loaded into the token. Also, Gen2 tokens are available in three physical sizes, the largest of which is like the current Gen1 tokens.
- Gen2 tokens have a 64-bit SafeNet driver for Linux. At the time of writing, the SafeNet driver for Gen1 tokens remains a 32-bit module, which is why current zPDT Gen1 token users must ensure that the 32-bit version of the `libstdc++`, `lib32stdc++6`, or `libstdc++6-32bit` module is installed on the base Linux system. (The exact module "short name" varies with different distributions and might change again after publication of this edition.)

[1] For this publication, we continue to use the product line name SafeNet as a more specific name instead of the owning companies, Gemalto, and now Thales (which acquired Gemalto).

- A special Gen2 “software license” is available, although it is limited to unique IBM ZD&T licenses at the time of writing. This license replaces the need for physical 1091 tokens.
  - This license is a Gen2 license that requires unique 64-bit SafeNet drivers and clients. It does not need the 32-bit `libstdc++` module.
  - Usage requires a separate personal computer (PC) license server. The Gen2 license cannot be used on the same Linux base running IBM ZD&T. (There might be special cases involving containers.) IBM ZD&T documentation should be obtained for more information. (Information Technology Company (ITC) can provide a tiny license server machine that is configured especially for this function.)
- The TCP/IP port number for a 1090 or 1091 license server is 9450; for a Gen2 license server it is 1947; and for a Unique Identity Manager (UIM) server it is 9451. These port numbers are not configurable.[2] If you *must* change one of these port numbers, you should contact your zPDT provider for assistance. Firewalls between the servers and clients must allow the required IP and port access for both TCP and UDP.
- More information about Gen2 tokens will be available before any usage of that version becomes necessary for zPDT users.
- Regardless of the type of token (or software license) that is used, an operational zPDT system identifies itself as an IBM type 1090 system.

Gen2 client configuration

You must complete two steps for Gen2 client operation (including the software-only license):

1. Activate the Gen2 software.
2. Configure your specific server information.

After the basic zPDT package (limited to IBM ZD&T for the software-only license) is installed, the Gen2 client can be activated.[3] Working as root, issue the following command:

```
/usr/z1090/bin/gen2_init
```

The resulting display depends on your Linux distribution, but it might look like the following lines:

```
# /usr/z1090/bin/gen2_init
  Installing LDK client side license manager ....
Preparing...                      ################################# [100%]
Updating / installing...
   1:aksusbd-7.40-1               ################################# [100%]
redirecting to systemctl start aksusbd.service
..Done.
```

This setup is done only once. Thereafter, the Gen2 client starts automatically when the client Linux system starts.

[2] This change is a new one for zPDT GA7.

[3] Earlier zPDT versions of the Gen2 client (and server) required a 32-bit version of the Linux glibc library and the Gen2 client installation process automatically accessed several Internet sites to obtain this library if it was not in the Linux system. It was necessary to have a working internet connection before starting the process to install the client or server.

Next, you must tell your client where to find the server by using the **clientconfig** command. If this command finds the Gen2 client software active, it displays the following lines:

```
Gen2 Server..............______________      (Must be specified.)
Gen2 Backup Server.......______________      (optional)
Gen1 ContactServer........localhost          (used if thee Gen2 server fails))
Gen1 BackupServer.........________________
UIM ContactServer.........________________
UIM Local Serial Random..._                  (y or blank)
Factory Reset............_                   (Enter "y" to reset file.)
```

You must specify at least one server. You can specify both Gen2 and Gen1 servers, but this situation would be unusual.

The specification order for a UIM server is the UIM ContactServer, the Gen2 Server, and then the Gen1 server. The first specified address in this order is used. Backup license server addresses are not considered for a UIM server. When installing a Gen2 server, you normally leave the UIM `ContactServer` field blank.

If the `Gen1 ContactServer` field is left blank, it internally defaults to `localhost`.

An alternative method for setting the **clientconfig** values for Gen2 licenses is described in 4.2.28, "The clientconfig_cli command" on page 90 or 4.2.40, "The ldk_server_config command" on page 98.[4] These alternative methods provide a command-line interface (CLI) that can be used within a Linux script.

The **query_license** command can be used with Gen2 clients and servers. It is useful for verifying your setup and connectivity with the license manager.

SafeNet provides a browser-based *Sentinel Admin Control Center* that can be used to configure the Gen2 functions. As a best practice, use the **clientconfig** and **serverconfig** commands rather than this browser interface.

The Gen2 license server (packaged with the standard UIM server) is *not* part of the standard IBM ZD&T package. A package with these two components is available as a separate deliverable. The separate package providing a Gen2 license server also contains a UIM server. The UIM server is the same in both cases.

Gen2 license server

Several steps are involved in preparing a Gen2 license server. The license server (and the associated UIM server) are supplied in a file with a name like that shown in the following command. Place this file in a convenient directory and working as root, run the file:[5]

```
# ./zPDT_LS-1-10.55.07.x86_64           (Use your correct file name.)
```

Both the Gen2 license server and a UIM server are installed.[6]

[4] The **ldk_server_config** command is deprecated. The **clientconfig_cli** command is its recommended replacement.

[5] The file must be an executable one, and it might require a **chmod u+x** operation. Also, the exact file name might change slightly to match newer levels of zPDT.

[6] The Gen2 server is installed in `/opt/IBM` instead of the traditional `/user/z1090/bin` that was used for other zPDT modules.

If you are installing software-only licenses (no Gen2 tokens), the next step is to obtain software licenses that can be "served" by the license server. Working as root, run the following command:

```
# /opt/IBM/LDK/request_license
```

This command creates a file that is named `<hostname>_xxxxxx.zip` in root's home directory, where <hostname> is your Linux system's name and xxxxxx is a timestamp. This file contains a *fingerprint* of the license server. You must send this file to the appropriate IBM ZD&T licensing facility (as identified by your IBM ZD&T contract). In return, you receive an `update_zip` file that contains the number and type of licenses that your server can supply to clients. Place this file into a convenient directory and install it as follows:

```
# /opt/IBM/LDK/update_license <hostname>_xxxxxx_update.zip
```

Restart the license server daemon with one of the following commands:

```
# systemctl restart aksusbd.service     (used with newer Linux distributions)
# service aksusbd restart               (used with older Linux distributions)
```

The Gen2 license server installation is complete. You also must start the UIM server on your server system. You should consider security controls for your Gen2 server.

The `update_zip` file that conveys IBM ZD&T licenses to the server also contains Application Development Controlled Distribution (ADCD) decryption licenses that become available to the client systems.

Important: You cannot reinstall a Gen2 software license file. You cannot "start over" with it. If you need to start over, you must run the **`request_license`** function again and obtain a new `update_license` file from your IBM ZD&T provider.

Managing the Gen2 server

After a Gen2 server is installed, the **`serverconfig`** and **`query_license`** commands can be used to help manage it. The **`serverconfig`** command, which is used on the Linux system that is running the Gen2 server, produces the following output:

```
Allow Server Access _                   (Enter Y or N.)
Enable Access Log   _                   (Enter Y or N.)

enter="Process, save-quit"   ESC="quit"
Rules will be loaded from /opt/IBM/LDK/rules.ini on save-quit
```

The first option, with an `N` operand, disables the Gen2 server without stopping it. When disabled, Gen2 server does not respond to client zPDT heartbeats (effectively stopping the clients), and does not issue new licenses. A `Y` operand starts normal server operation again. The **`Log`** option causes the server to create an internal log that is automatically trimmed every 60 days. (The default is no log.) You can obtain a copy of the current log with the **`display_gen2_acclog`** command.

The **`serverconfig`** command forces rereading of the Gen2 server security file.

The **`query_license`** command can be used on both Gen2 clients and Gen2 servers. (It does not handle Gen1 servers or clients.) For a client, it displays the details about the licenses in use and the server that is being used. For a server, it displays the inventory of licenses that are available and information about all current users.

Here is the command syntax and output:

```
$ query_license
The following key is available:
HASP-SL key=id=367116869417668380 features:
FID  Feature Name        Expiration            Logins  MaxLogins
334- ADCD License    Thu Jan 22, 2017 19:59:59   0        1
335-  CPU License    Thu Jan 22, 2017 19:59:59   1       12

Host Information: Falcon 42 localhost
Z109x detected. Only client sessions will be shown
KeyID   FID  FeatureName   Address     user       Machine        Login
3671..  335  CPU license 9.56.111.222 ibmsys1 bill.privx.ibm.com Wed Feb 15
```

The syntax is slightly different than in other parts of zPDT. `Logons` and `MaxLogins` refer to licenses, not users. A user with three CPs would have three “logins.” In this example, the server has 12 zPDT CPU licenses, and only one of them is being used. Information about the user’s machine is shown.

If you want to move your Gen2 server to a different PC, you must contact your zPDT provider. They have a process for deleting the current Gen2 licenses and enabling a move to a new server. Alternatively, the IBM ZD&T service can provide a different way to move the license server to a different physical machine in certain situations. You cannot move the server files (or the hard disk drive (HDD) containing the files) to a different machine.

The `man` files for a Gen2 license server are in `/opt/man/IBM/LDK/man1`. To access them, you must include this directory in the `man` path for root. For example:

```
export MANPATH=$MANPATH:/opt/man/IBM/LDK
```

The Gen2 client function can be removed by running the following command:

```
# /usr/z1090/bin/gen2_init  --remove       (note the two dashes)
```

A Gen2 server is a package that can be removed with a command such as the following one (where the exact file name should match whatever name was used to install the Gen2 server function):

```
# ./zPDT_LS-1.8.51.10-x86_64 --remove
```

This command also automatically removes the UIM server that was associated with the Gen2 server.

Do not use **rpm** commands to remove these functions because it will not remove all the necessary modules.

Important: License and UIM server port numbers (9450 and 9451 for Gen1, and 1947 and 9451 for Gen2) must not be used by any other IP function on your subnet. A port number conflict can cause zPDT failure.

A Gen2 server inspects th file `/opt/IBM/LDK/rules.ini` for security rules. This file is a text file that you manage with your Linux text editor. The file should contain lines like the following ones:

```
allow=192.168.1.*
allow=my.favorite.domain
deny=all
```

The Gen2 server reads from the top of the file and stops when the client (who is requesting a license) matches a rule. The first rule that matches the client's IP address is used. Later rules are not inspected. The Gen2 server automatically appends `allow=all` to the bottom of the list.

In this example, any client on subnet `192.168.1` or the subnet (or host) that is resolved by `my.favorite.domain` can obtain a license from the server (if any are available). Everyone else is rejected.

You can change the `rules.ini` file while the server is running. To cause the Gen2 server to reload the rules file, run the following command:

```
# serverconfig -u
```

The commands to restart the Gen2 server are as follows:

```
# systemctl restart aksusbd.service      (newer Linux distributions)
# service aksusbd restart                (older Linux distributions)
```

Gen2 new licenses

License renewal is done by obtaining a new update file from your zPDT provider.

Place this file into a convenient directory and install it by running the following command:

```
# /opt/IBM/LDK/update_license  <hostname>_xxxxxx_update.zip
```

This command is the same one that was used to install the initial Gen2 license enablement file.

Dedicated key server

ITC[7] offers an optional zKey appliance that functions as a stand-alone software key server. The zKey appliance is the small box sitting in front of the server, as shown in Figure C-1 on page 385. The zKey has a single system board computer running Linux, with 4 GB of memory and a 64 GB solid-state drive (SSD) drive. zKey has a display port, a USB-2 port, a USB-3 port, and two local area network (LAN) 10/100/1000 ports. A user-provided display, mouse, and keyboard are connected when configuring the zKey, but they are not needed for later operation.

The internal Linux (in the zKey) is Ubuntu, and normal Linux commands are available for customization. The zPDT commands that are associated with a Gen2 software license server are available, and the license installation and operation is the same as described in "Gen2 license server" on page 381. As an additional option, the web-oriented tools of the IBM ZD&T Enterprise Edition also can be installed in the zKey.

Although the zKey offers no functional advantage over a "normal" PC that is used as a license server, it has practical advantages in size, operational stability, and the absence of external controls (after customization) that tempt curious fingers. The intention is a "fire and forget" appliance. For more information, contact ITC.

[7] Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042, or sales@itconline.com.

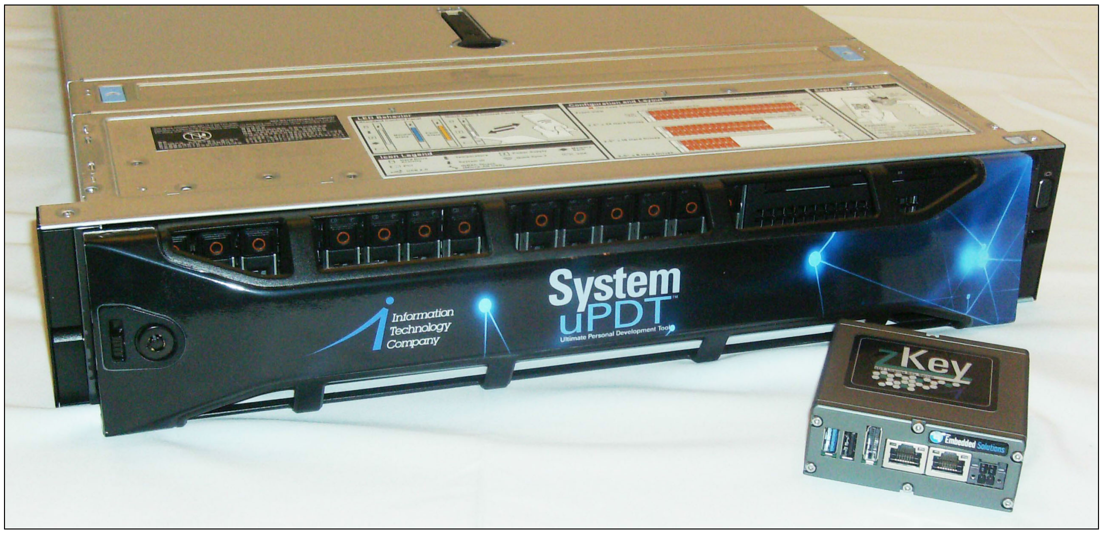

Information Technology Company
System uPDT
Ultimate Personal Development Tools
zKey
Embedded Solutions

Figure C-1 zKey (and uPDT Enterprise Server)

As a side note, the server in Figure C-1 (an ITC uPDT Enterprise Server) was, at the time of writing, the fastest zPDT machine that we used. If you are interested in pushing the boundaries of zPDT performance, consider looking at it.

D

IBM Z Development and Test Environment notes

This publication is primarily about Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) and is not intended to cover IBM Z Development and Test Environment (ZD&T) details. While much of the content might also apply to IBM ZD&T, the text does not cover distinctions that are unique to IBM ZD&T. This appendix contains only a few details about IBM ZD&T. Any IBM ZD&T user (or potential user) should request access to documentation from their IBM ZD&T vendor.

The ISV zPDT package that is available to independent software vendors (ISVs) and IBM internal users requires a 1090 token (or a remote license manager with an equivalent license). It cannot function with a 1091 token. IBM ZD&T systems use 1091 tokens or a remote software license manager and cannot function with a 1090 token. Users of IBM ZD&T who require the Coupling Facility (CF) must purchase license features that enable that function.

The license terms and conditions for the two packages are different and not addressed in this publication. For complete licensing terms, contact your IBM representative, your IBM ZD&T supplier, or refer to the license documents that are supplied with your copy of IBM ZD&T.

Table D-1 summarizes some of the basic differences between ISV zPDT and IBM ZD&T.

Table D-1 1090 and 1091 comparisons

ISV zPDT usage: 1090 tokens	IBM ZD&T usage: 1091 tokens or equivalent
1090 token only. 1091 token is not usable (or equivalent remote licenses).	1091 token only. 1090 token is not usable (or equivalent remote licenses or a Gen2 software license).
Maximum of 8 CPs (with multiple tokens).	Maximum of 8 CPs (with multiple tokens or a “large” token or software license).
CF usage (under z/VM).	CF usage (under z/VM) only with an additional license feature.

ISV zPDT usage: 1090 tokens	IBM ZD&T usage: 1091 tokens or equivalent
1, 2, or 3 licenses in token. No way to order larger tokens.	Tokens with various numbers of licenses are available.
Installed rpm or deb name is z1090.	Installed rpm or deb name is z1091.
ISV zPDT and zD&T cannot be installed on the same machine.	
"Standard" z/OS Application Development Controlled Distribution (ADCD) system.	Might have additions to the z/OS ADCD system, with possible delay for the most current z/OS.
Most zPDT commands are the same. A few have 1090 or 1091 versions.	
Functional modules are installed in `/usr/z1090/bin` with additional materials in `/usr/z1090/man` and `/usr/z1090/uim`. zPDT instance files are in a subdirectory that is named `z1090` in the Linux zPDT user ID home directory. There is no `/usr/z1091` or `~/z1091` usage.	
z/VM available (with proper license).	z/VM available for limited use only for use with the optional CF license.
IBM z/Transaction Processing Facility (z/TPF) not available at the time of writing.	z/TPF availability can be requested.
Installation and operational commands are as described in this publication. (Information Technology Company (ITC) offers additional interfaces with their uPDT package).	IBM ZD&T provides many additional installation and operation programs and interfaces that are not described in this publication.

The **cpuopt** statement in a device map (devmap) specifies optional parameters for the CPs. The following statement should be used only by IBM ZD&T customers who have the optional CF feature with their license:

```
cpuopt zVM_CouplingFacility          (no blanks in operand)
cpuopt zVM_Coupling                  (This abbreviation can be used.)
```

The zVM_CouplingFacility operand is significant only for zD&T systems, which must have the proper license feature to enable it. In effect, the **zVM_CouplingFacility** function is always present for ISV zPDT systems.

Important: IBM ZD&T users *must* specify **zVM_Coupling** if their license has this feature. If the feature is present in the license and is not in the devmap, IBM ZD&T fails to acquire the licenses.

Tip: The CPU "model" number (which can be displayed on the MVS console or through some specialized IBM zSystems system instructions) is always 1090 regardless of whether ISV 1090 (token 1090) or some form of IBM ZD&T (token 1091 or software license) is used.

Here is another devmap statement:

```
[system]
...
rdtserver 27000@our.server.acme.com    # Rational License Server and port
```

The **rdtserver** statement is used only with an IBM ZD&T system. It points to an IBM Rational License Server that is used to supplement the IBM ZD&T license.[1] A Rational License Server is not the same as an IBM ZD&T remote license server, and it is not required for basic IBM ZD&T operation. The operand can be a normal URL domain name or an absolute IP numeric address. Also, note the format with the port number placed before the address. Multiple servers can be specified by using a colon as the separator:

```
rdtserver 27000@server1.com:7777@server2.com
```

The “software only” licenses that are available for IBM ZD&T are considered Gen2 licenses and have a number of different installation and update options. They are briefly described in Appendix C, “Generation 2 tokens and licenses” on page 379. The “software only” licenses are not available for ISV zPDT systems.

ISV zPDT (1090 tokens) and IBM ZD&T (1091 tokens) have different procedures for obtaining token licenses. Here are the differences For IBM ZD&T (1091 tokens):

- The license provider is aware of token license expiration dates (or a request is made to the license provider). No `.req` file is involved.
- The provider sends a compressed file that is installed with the **`Z1091_token_update`** command. Do *not* decompress the file.
- Especially for IBM ZD&T users with physical tokens, when you request a license file, ensure that you provide the correct token serial number. It is easy to misread the numbers on the token. With an incorrect token number, you received a license file and it appears to install on the token, but it does not work.

[1] For more information about Rational licenses, contact your IBM marketing representative.

Secure x3270 connection

This section describes steps that might be used to provide a secure connection capability for a z/OS system that is not configured for it. These steps are *not* needed for Application Development Controlled Distribution (ADCD) 2.4 and later. These releases are configured for secure connections through port 2023. However, the following details can be helpful for z/OS systems that are not configured for secure connections or for someone who needs to change their setup.

Tip: This material is intended only for Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT) (ISV zPDT) users who have specific requirements for more secure 3270 terminal operation; who find that the secure options in ADCD z/OS 2.4 (and later) are not appropriate; and who have some understanding of the various parameters and operations that are described in this section.

The steps that are described in this section were valid with ADCD z/OS 2.4. Later ADCD z/OS releases *might* require some alterations. Some of this material is specific to the x3270 terminal emulator, which is a non-IBM open source program. Other 3270 terminal emulators might have different requirements.

The following material was contributed by Marc Van der Meer and Paul Mattes.

The basic mechanism requires TN3270[1] parameters that define a port number for a secure connection and specify the certificate to be used. The certificate provides a public key that is sent to the client (x3270) during session setup negotiations. Then, the client uses the public key to encrypt the session setup details, including the session key to be used. This session key provides an encrypted 3270 session, but the client cannot be certain it is connected to the correct host.

1 TN3270 is the z/OS started task that provides 3270 operation through TCP/IP.

The client can verify that the name in the certificate that is presented by the host matches the expected name. By default, x3270 3.6 and later verify host certificates, but earlier releases do not do this task by default. More x3270 parameters can be used to force verification (for x3270 releases earlier than 3.6), disable verification (for x3270 3.6 and later), and override the hostname being checked. At the time of writing, we have ISV zPDT users with both earlier and later releases. You can verify your x3270 level with the command **`x3270 --version`**.

We used the following steps in our example:

1. Change the z/OS hostname. This step is not required.
2. Create the necessary certificates by using the **`RACDCERT`** utility.
3. Update the parameters in the TN3270 configuration that is used by z/OS TCP/IP.
4. Export the certificate authority (CA) certificate to the system that provides the x3270 sessions. This system can be your base Linux or an external Linux system. This step is needed only if host verification is done.
5. Update Linux the hostname resolution files, if needed.
6. Start x3270 with the appropriate parameters.

Changing the z/OS hostname

Recent ADCD z/OS systems have `S0W1.DAL-EBIS.IHOST.COM` as the default hostname. You can continue to use it or you can change it.[2] (You can use the **`hostname`** command within OMVS to display your hostname.) The following text configures our hostname as part of the certificate as is needed for host verification.

We use the z/OS hostname `S0W1.OGDEN.IBM`, which is an arbitrary name that probably will not be recognized by an internet name server. We changed the z/OS hostname by editing the following items:[3]

- For `ADCD.Z23D.TCPPARMS(TCPDATA)`, change the **`DOMAINORIGIN`** parameter to your selection. We changed it to `OGDEN.IBM`. We left the **`HOSTNAME`** statement as `S0W1`.[4]
- **`ADCD.Z23D.TCPPARMS(GBLIPNOD)`** was changed as follows. (We assume that this statement is primarily for documentation and that you might have more statements in this member.)

  ```
  10.1.1.2 S0W1.OGDEN.IBM
  ```
- `ADCD.Z23D.TCPPARMS(GBLTDATA)` has the same structure as the `TCPDATA` member, and we made the same changes to it.
- Although `IZUPRMxx` is not related to the secure x3270 sessions, we also changed the full hostname in the `IZUPRMxx` members in `PARMLIB`.

Stop and restart z/OS TCP/IP and then try the OMVS **`hostname`** command to verify your changes.

[2] If your z/OS is directly connected to the internet with a routable IP address and you want to use a domain name that correctly routes to your system, work with your network team to create an appropriate hostname.

[3] Some of the data set names that are mentioned here have `Z23D` as a middle qualifier in the name. This qualifier changes with different z/OS ADCD releases, so you must adjust the instructions.

[4] These specific z/OS data set names correspond to the ADCD z/OS 2.3 release. You must adapt the names to your current system.

Creating certificates

At the time of writing, the ADCD z/OS system lets the TN3270 start task default to user ID START1.[5] Create and run the following job, and change START1 and S0W1.OGDEN.IBM as required for your system. You must have RACF SPECIAL authority or the appropriate PERMIT.

```
//CERT1 JOB 1,OGDEN,MSGCLASS=X
//CRECERTS EXEC PGM=IKJEFT1A
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
SETROPTS CLASSACT(DIGTCERT) RACLIST(DIGTCERT)

RACDCERT GENCERT CERTAUTH  +
 SUBJECTSDN(CN('TN3270_signer')) +
 WITHLABEL('TN3270_signerCA') +
 NOTAFTER(DATE(2029/12/31)) SIZE(2048)
SETROPTS RACLIST(DIGTCERT) REFRESH

RACDCERT GENCERT ID(START1) +
 SUBJECTSDN(CN('S0W1.OGDEN.IBM')) +
 WITHLABEL('TN3270TLS') +
 SIGNWITH(CERTAUTH LABEL('TN3270_signerCA')) +
 NOTAFTER(DATE(2029/12/31)) SIZE(2048)
SETROPTS RACLIST(DIGTCERT) REFRESH

RACDCERT ADDRING(TN3270TLSring) ID(START1)
SETROPTS RACLIST(DIGTCERT) REFRESH

RACDCERT CONNECT(CERTAUTH LABEL('TN3270_signerCA') +
 RING(TN3270TLSring) CERTAUTH) ID(START1)
SETROPTS RACLIST(DIGTCERT) REFRESH

RACDCERT CONNECT(ID(START1) LABEL('TN3270TLS') +
 RING(TN3270TLSring) DEFAULT) ID(START1)
SETROPTS RACLIST(DIGTCERT) REFRESH

RACDCERT EXPORT(LABEL('TN3270_signerCA')) CERTAUTH +
 DSN('OGDEN.CERT.B64') FORMAT(CERTB64)
/*
```

A few comments about this job:

- You can perform these functions through Time Sharing Option (TSO) instead of using a batch job, but the statements ("commands") are a bit complex, and keying entries should be avoided.
- The first **RACDCERT** statement creates a self-signed CA certificate that is stored in the RACF database. The SUBJECTSDN is arbitrary, that is, a name must be specified, but it is not used for any purpose in our example.
- The **ID(START1)** parameter is the z/OS user ID that is associated with the TN3270 address space.

[5] You can verify this configuration by examining the TN3270 Job Entry Subsystem (JES) log by using System Display and Search Facility (SDSF).

- The second **RACDCERT** statement creates a certificate that is signed by the CA we created that will be used by TN3270 to set up the encrypted sessions. SUBJECTSDN is the hostname of the z/OS system. This hostname is relevant for client (x3270) validation of the certificate. Even if your x3270 level does not default to certificate validation, as a best practice, specify the hostname correctly to allow for future x3270 updates.
- The next three **RACDCERT** statements create a key ring (in RACF) containing the two certificates that we created.
- The last **RACDCERT** statement exports a version of the CA certificate that is needed for host validation. In practice, we sent a copy of this file to our base Linux and to other Linux systems that might be used to connect to our z/OS. The data set that was created (OGDEN.CERT.B64 in the example) is RECFM=VB, LRECL=84. We used the x3270 File Copy function to copy it (in ASCII, not binary) to a Linux file on our base Linux. We copied the file to cert.pem in the Linux directory that we used for starting ISV zPDT. The B64 suffix name (for the z/OS data set) and the pem suffix (for the Linux file) follow the standard naming convention for such files.

Updating local name resolution

On our base Linux, we edited /etc/hosts to include the following lines:

```
127.0.0.1   localhost
10.1.1.2    SOW1.OGDEN.IBM
```

Updating TN3270 parameters

We decided to use TCP port 2023 for secure TN3270 sessions. This choice is not arbitrary because we observed that this port was partly set up this way in some older ADCD z/OS systems and on other z/OS systems. You must inspect and update a few TN3270 parameters for port 2023. The relevant parameters are in ADCD.Z23D.TCPPARMS starting in member TN3270. Examining this member, we saw that it included members PORT0023 and PORT2023. Using this ADCD TN3270 structure, we need to update parameters in member PORT2023.

This member contains several statements in a TelnetParms...EndTelnetParms stanza. Change this stanza to read as follows:

```
TelnetParms
  ConnType Secure
  Secureport 2023
  Keyring SAF TN3270TLSring     ;this is the keyring that you created
  ClientAuth NONE               ;Server side only authentication
  NOSSLV2
  NOSSLV3                       ;Force TLS
  Encryption
    SSL_AES_256_SHA             ;Strong one!
  EndEncryption
  SSLtimeout 5
EndTelnetParms
```

Then, we used the z/OS commands **P TN3270** and **S TN3270** to restart TN3270.

Making the connection

The x3270 program adds a prefix to (a leading character) some operands. Note the `L:` and `Y:` prefixes in the following examples. The `L:` prefix indicates that a secure session is requested and the `Y:` prefix disables host certificate validation. We have several options for creating a secure connection:

- With x3270 3.5 or earlier:
  - No host certificate validation:
    - `x3270 L:SOW1.OGDEN.IBM:2023`
    - `x3270 L:10.1.1.2:2023` (`10.1.1.2` is the IP address of z/OS TCP/IP.)
  - With host certificate validation:
    - `x3270 -verifycert -cafile cert.pem L:SOW1.OGDEN.IBM:2023`
    - `x3270 -verifycert -cafile cert.pem -accepthostname DNS:SOW1.OGDEN.IBM L:10.1.1.2:2023`
- With x3270 3.6 or later:
  - No host certificate validation:
    - `x3270 L:Y:SOW1.OGDEN.IBM:2023`
    - `x3270 L:Y:10.1.1.2:20233`
  - With host certificate validation:
    - `x3270 -cafile cert.pem L:SOW1.OGDEN.IBM:2023`
    - `x3270 -cafile cert.pem -accepthostname DNS:SOW1.OGDEN.IBM L:10.1.1.2:2023`

These examples assume that file `cert.pem` is in the current Linux directory, but if not, a full Linux path name is needed.

Additional notes

The **`-accepthostname`** option is used if you want host certificate validation and `SUBJECTSDN` *`CN attribute`* in the host certificate does not match the server's hostname:

```
-accepthostname DNS:SOW1.OGDEN.IBM (note the DNS prefix)
```

The `L:` prefix that is available with x3270, as in `x3270 L:10.1.1.2`, can be confusing. The best explanation is that if the host is set up for a secure connection, use the `L:` prefix, otherwise do not. The usage is optional, but it speeds the initial session negotiations.

If the host is set up for secure connections and you do not specify a certificate (by using the **`-cafile`** parameter, for example), and you are using host verification, then the host begins a secure connection and x3270 attempts to use the root CA certificates that are built in to the base Linux system.

Should every user be on the latest level of x3270? The realistic answer is that this situation is not likely. The complete x3270 suite is complex and contains more than the basic x3270 terminal emulator. Building an instance of x3270 involves compile options and library requirements. In practice, the groups providing the common Linux distributions each produce their own "build" of x3270, which tend to be at different release and library levels. If you use commands such as **`yum install x3270`** or **`zypper install x3270`** to obtain an x3270 for your Linux distribution, you obtain whatever release was built for that Linux distribution. You can build your own x3270 from source, but this process is complex.

Reworking your certificates

The following commands might be useful if you want to rework your certificates. (RACF SPECIAL authority is needed.)

```
RACDCERT LIST ID(START1)
RACDCERT LIST(LABEL('TN3270TLS')) ID(START1)
RACDCERT LISTRING(TN3270TLSring) ID(START1)
RACDCERT LIST ID(START1)
RACDCERT DELRING(TN3270TLSring) ID(START1)
RACDCERT DELETE(LABEL('TN3270_signerCA')) CERTAUTH
RACDCERT DELETE(LABEL('TN3270TLS')) ID(START1)
SETROPTS RACLIST(DIGTCERT) REFRESH
```

Some names are enclosed in quotation marks and some are not enclosed. In these examples, the ID value and the certificate and ring names are variable to match your definitions.

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this publication.

IBM Redbooks

The following IBM Redbooks publication provides more information about the topics in this publication:

- *zPDT Sysplex Extensions - 2020*, SG24-8386

You can search for, view, download, or order this document and other Redbooks, Redpapers, web docs, drafts and, additional materials, at the following website:

ibm.com/redbooks

Other publications

This publication also is relevant as a further information source:

- *z/Architecture Principles of Operation,* SA22-7832

Online resources

This website also is relevant as a further information source:

- An informal internet forum is often used for discussions about ISV zPDT. (This forum is not sponsored by IBM.)

  https://groups.io/g/zPDT

Abbreviations and acronyms

ACP	Access Control Point
ADCD	Application Development Controlled Distribution
AP	adjunct processor
ASID	address space ID
AVR	automatic volume recognition
BCPii	Base Control Program internal interface
BLP	Bypass Label Processing
CA	certificate authority
CCA	Common Cryptographic Architecture
CCW	channel command word
CF	Coupling Facility
CFCC	Coupling Facility Control Code
CLI	command-line interface
CMS	Conversational Monitor System
COD	Customized Offering Driver
CPACF	CP Assist for Cryptographic Functions
CSF	Cryptographic Service Facility
CTC	channel-to-channel
CTN	Coordinated Timing Network
DCB	dataset control block
devmap	device map
DLT	digital linear tape
DSN	data set name
EAV	extended address volume
EBCDIC	Extended Binary Coded Decimal Interchange Code
EIO	emulated I/O
FB	Fixed Block
FBA	Fixed Block Architecture
FCB	Forms Control Buffer
FCP	Fibre Channel Protocol
GRS	global resource serialization
GVRP	Generic VLAN Registration Protocol
HDD	hard disk drive
HIS	Hardware Instrumentation Services
HLQ	High-Level Qualifier
HMC	Hardware Management Console
HSA	Hardware System Area
IBM	International Business Machines Corporation
IBM ZD&T	IBM Z Development and Test Environment
ICC	Integrated Console Controller
ICSF	Integrated Cryptographic Service Facility
IFL	IBM Integrated Facility for Linux
IMS	IBM Information Management System
IODF	input/output definition file
IPL	initial program load
ISFC	Inter-System Facility for Communications
ISPF	IBM Interactive System Productivity Facility
ISV	independent software vendor
ISV zPDT	Independent Software Vendor (ISV) IBM Z Program Development Tool (IBM zPDT)
ITC	Information Technology Company
JES	Job Entry Subsystem
JIT	Just In Time
KVM	Kernel-based Virtual Machine
LAN	local area network
LCS	LAN Channel Station
LPAR	logical partition
MI	machine interface
MIH	Missing Interrupt Handler
MIPS	million instructions per second
MT	multithreading
MTU	maximum transmission unit
NAS	network-attached storage
NAT	Network Address Translation
NJE	Network Job Entry
NL	New Line
NTP	Network Time Protocol
OAT	OSA Address Table
OSA	Open Systems Adapter
OSA/SF	Open Systems Adapter/Support Facility

OSD	OSA-Express Direct
OSE	OSA-Express
PAV	Parallel Access Volumes
PC	personal computer
PDS	partitioned data set
PDSE	partitioned data set extended
PID	process ID
PSW	Program Status Word
PTF	program temporary fix
PU	processing unit
PUT	Program Update Tape
RAS	reliability, availability, and serviceability
RHEL	Red Hat Enterprise Linux
RSCS	Remote Spooling Communications Subsystem
RSU	Recommended Service Upgrade
SAD	stand-alone dump
SDSF	System Display and Search Facility
SEL	Security Enhanced Linux
SIE	START INTERPRETIVE EXECUTION
SMT	symmetric multithreading
SNA	Systems Network Architecture
SRDI	Security-Relevant Data Items
SSC	IBM Secure Service Container
SSCH	Start Subchannel
SSD	solid-state drive
SSI	Single System Image
STC	Started Class
STIDP	Store CPU ID
STP	Server Time Protocol
TAI	International Atomic Time
TKE	Trusted Key Entry
TOD	time-of-day
TSAF	Transparent Services Access Facility
TSO	Time Sharing Option
UCS	Universal Character Set
UDX	user-defined extension
UIM	Unique Identity Manager
UNLABB	unlabeled tape
UTC	Coordinated Universal Time
VLAN	virtual LAN
VM	virtual machine
VMAC	virtual MAC
VPD	Vital Product Data
VSAM	Virtual Storage Access Method
VTAM	IBM Virtual Telecommunications Access Method
VTOC	Volume Table of Contents
VVDS	VSAM Volume Data Set
WLM	workload manager
WTM	Write Tape Mark
WTO	Write To Operator
XCF	Cross-System Coupling Faciity
z/TPF	IBM z/Transaction Processing Facility
zAAP	IBM zSeries Application Assist Processor
zBX	IBM zEnterprise BladeCenter Extension
zCX	IBM z/OS Container Extensions
zEDC	iBM zEnterprise Data Compression
zIIP	IBM zSystems Integrated Information Processor

Index

Redbooks

ISV IBM zPDT Guide and Reference

SG24-8205-06

ISBN 0738460834

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8205-06

ISBN 0738460834

Printed in U.S.A.