

# IBM zPDT Guide and Reference

## System z Personal Development Tool

IBM System z Personal Development Tool

Full IBM z/OS usage

Linux base



Bill Ogden

**Redbooks**





International Technical Support Organization

**IBM zPDT Reference and Guide: System z Personal  
Development Tool**

March 2020

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

#### **Fifth Edition (March 2020)**

This edition applies to Version 1, Release 10, of the IBM System z Personal Development Tool (zPDT).

# Contents

<b>Notices</b> .....	xi
Trademarks .....	xii
 <b>Preface</b> .....	 xiii
Authors .....	xiii
Now you can become a published author, too! .....	xiv
Comments welcome .....	xiv
Stay connected to IBM Redbooks .....	xiv
 <b>Chapter 1. Introduction</b> .....	 1
1.1 General architecture .....	1
1.2 zPDT security, integrity, and RAS concepts .....	3
1.3 Terminology .....	4
1.4 PC choices .....	5
1.5 Installation and operation .....	6
 <b>Chapter 2. Function, releases, content</b> .....	 7
2.1 System z characteristics .....	8
2.1.1 Architecture levels .....	9
2.2 Hardware token .....	9
2.2.1 Emulated I/O .....	11
2.2.2 Concurrent PC workloads .....	12
2.3 Operational overview .....	13
2.3.1 Linux userids .....	13
2.3.2 zPDT instances .....	13
2.3.3 zPDT console .....	14
2.3.4 Performance .....	14
2.4 Base configurations .....	16
2.4.1 Hardware and software levels .....	16
2.5 Using older System z architectures .....	18
2.6 zPDT Components .....	18
2.6.1 zPDT operational elements .....	19
2.6.2 Memory .....	19
2.6.3 Disk space .....	20
2.6.4 LAN adapters .....	20
2.6.5 Device maps .....	21
2.6.6 Linux directory structure .....	22
2.6.7 zPDT control structure .....	23
2.7 ISV zPDT and zD&T differences .....	23
2.8 zPDT releases .....	24
2.8.1 Current release (GA10) .....	24
2.9 Previous zPDT releases .....	25
 <b>Chapter 3. Devmaps</b> .....	 33
3.1 Device maps .....	33
3.2 System stanza .....	34
3.2.1 Adjunct-processor stanza .....	38
3.2.2 System timer protocol stanza .....	39
3.2.3 zEDC stanza .....	39

3.3 Manager stanzas. . . . .	39
3.3.1 The awsckd device manager . . . . .	40
3.3.2 The awsfba device manager. . . . .	41
3.3.3 The aws3274 device manager . . . . .	42
3.3.4 The awstape device manager. . . . .	44
3.3.5 The awsosa device manager . . . . .	45
3.3.6 The awsrdr device manager . . . . .	46
3.3.7 The awsprt device manager . . . . .	47
3.3.8 The awscmd device manager. . . . .	48
3.3.9 The awsscsi device manager . . . . .	49
3.3.10 The aws3215 device manager . . . . .	49
3.3.11 The awsoma device manager. . . . .	49
3.3.12 The awsctc device manager . . . . .	50
<b>Chapter 4. zPDT commands. . . . .</b>	<b>51</b>
4.1 The commands with examples . . . . .	52
4.1.1 The adstop command . . . . .	52
4.1.2 The alcckd command . . . . .	52
4.1.3 The alcfba command . . . . .	54
4.1.4 The ap_create command . . . . .	55
4.1.5 The ap_destroy command . . . . .	55
4.1.6 The ap_query command. . . . .	55
4.1.7 The ap_von and ap_voff commands. . . . .	55
4.1.8 The ap_vpd command . . . . .	56
4.1.9 The ap_zeroize command . . . . .	56
4.1.10 The attn command . . . . .	56
4.1.11 The aws_bashrc and aws_sysctl commands . . . . .	57
4.1.12 The aws_findlinuxtape command . . . . .	57
4.1.13 The aws_tapelnit command . . . . .	57
4.1.14 The aws_tapelnspl command . . . . .	58
4.1.15 The awsckmap command. . . . .	58
4.1.16 The awsin command. . . . .	58
4.1.17 The awsmount command . . . . .	59
4.1.18 The awsstart command. . . . .	60
4.1.19 The awsstat command . . . . .	61
4.1.20 The awsstop command. . . . .	62
4.1.21 The card2tape command . . . . .	62
4.1.22 The card2txt command . . . . .	63
4.1.23 The ckdPrint command. . . . .	63
4.1.24 The clientconfig command . . . . .	63
4.1.25 The clientconfig_authority command . . . . .	64
4.1.26 The clientconfig_cli command. . . . .	64
4.1.27 The cpu command . . . . .	65
4.1.28 The d command . . . . .	65
4.1.29 The display_gen2_acclog command. . . . .	67
4.1.30 The fbaPrint command . . . . .	67
4.1.31 The find_io command . . . . .	67
4.1.32 The gen2_init command . . . . .	68
4.1.33 The hckd2ckd and hfba2fba commands. . . . .	68
4.1.34 The interrupt command. . . . .	69
4.1.35 The ipl command . . . . .	70
4.1.36 The ipl_dvd command. . . . .	70
4.1.37 The ldk_server_config command . . . . .	71

4.1.38	The listVtoc command . . . . .	72
4.1.39	The loadparm command . . . . .	72
4.1.40	The managelogs command . . . . .	73
4.1.41	The memId command . . . . .	73
4.1.42	The mount_dvd command . . . . .	73
4.1.43	The msgInfo command . . . . .	74
4.1.44	The oprmsg command . . . . .	74
4.1.45	The pdsUtil command . . . . .	75
4.1.46	The query command . . . . .	76
4.1.47	The query_license command . . . . .	77
4.1.48	The rassummary command . . . . .	77
4.1.49	The ready command . . . . .	78
4.1.50	The request_license command . . . . .	78
4.1.51	The restart command . . . . .	78
4.1.52	The scsi2tape command . . . . .	79
4.1.53	The SecureUpdateUtility command . . . . .	80
4.1.54	The SecureUpdate_authority command . . . . .	80
4.1.55	The senderdata command . . . . .	81
4.1.56	The serverconfig command . . . . .	81
4.1.57	The serverconfig_cli command . . . . .	82
4.1.58	The settod command . . . . .	82
4.1.59	The snapdump command . . . . .	83
4.1.60	The st command . . . . .	83
4.1.61	The start command . . . . .	84
4.1.62	The stop command . . . . .	85
4.1.63	The storestatus command . . . . .	85
4.1.64	The storestop command . . . . .	86
4.1.65	The stpserverstart command . . . . .	86
4.1.66	The stpserverstop command . . . . .	86
4.1.67	The stpserverquery command . . . . .	86
4.1.68	The sys_reset command . . . . .	86
4.1.69	The tape2file command . . . . .	87
4.1.70	The tape2scsi command . . . . .	87
4.1.71	The tape2tape command . . . . .	88
4.1.72	The tapeCheck command . . . . .	88
4.1.73	The tapePrint command . . . . .	89
4.1.74	The token command . . . . .	89
4.1.75	The txt2card command . . . . .	90
4.1.76	The uimcheck command . . . . .	90
4.1.77	The uimreset command . . . . .	90
4.1.78	The uimserverstart command . . . . .	90
4.1.79	The uimserverstop command . . . . .	91
4.1.80	The update_license command . . . . .	91
4.1.81	The Z1090_ADCD_install and Z1091_ADCD_install commands . . . . .	91
4.1.82	The Z1090_token_update and Z1091_token_update commands . . . . .	92
4.1.83	The Z1090_removal command . . . . .	93
4.1.84	The z1090instcheck command . . . . .	93
4.1.85	The z1090term command . . . . .	93
4.1.86	The z1090ver and z1091ver command . . . . .	94
4.1.87	The zpdtSecureUpdate command . . . . .	94
<b>Chapter 5. zPDT installation . . . . .</b>		<b>95</b>
5.1	Linux Installation . . . . .	96

5.2 The zPDT program package . . . . .	97
5.3 zPDT installation . . . . .	97
5.3.1 Alter Linux files . . . . .	99
5.3.2 Other Linux notes . . . . .	101
5.4 x3270 . . . . .	101
5.4.1 x3270 keyboard maps. . . . .	102
5.5 Starting your new zPDT system . . . . .	102
5.6 Installing a different zPDT release . . . . .	103
<b>Chapter 6. AD-CD installation . . . . .</b>	<b>105</b>
6.1 General principles . . . . .	106
6.2 z System operating systems . . . . .	106
6.2.1 Media . . . . .	106
6.3 Installing a z/OS AD-CD system . . . . .	107
6.3.1 Specific installation instructions . . . . .	107
6.3.2 IODF device numbers . . . . .	108
6.3.3 zPDT control files . . . . .	109
6.3.4 IPL and operation . . . . .	110
6.3.5 Shutting down . . . . .	111
6.3.6 Startup messages . . . . .	111
6.3.7 Local volumes . . . . .	112
6.4 Multiple operating systems . . . . .	112
<b>Chapter 7. LANs . . . . .</b>	<b>115</b>
7.1 Overview of LAN usage . . . . .	115
7.1.1 zPDT 3270 interfaces . . . . .	116
7.2 OSA paths . . . . .	117
7.3 Scenarios . . . . .	119
7.4 Basic QDIO setup for z/OS . . . . .	119
7.5 Five scenarios . . . . .	120
7.5.1 Scenario 1 . . . . .	120
7.5.2 Scenario 2 . . . . .	122
7.5.3 Scenario 3 . . . . .	124
7.5.4 Scenario 4 . . . . .	126
7.5.5 Scenario 5 . . . . .	128
7.5.6 Scenario comparison . . . . .	129
7.5.7 z/OS resolver . . . . .	130
7.5.8 Local router LAN setups . . . . .	132
7.5.9 Investigating lan problems . . . . .	133
7.6 Performance problems . . . . .	133
7.6.1 Jumbo frames . . . . .	134
7.7 Local connections to z/OS TCPIP . . . . .	135
7.8 Choices . . . . .	136
7.9 Automatic configuration. . . . .	136
7.10 Useful z/OS networking commands . . . . .	137
7.11 More complete QDIO example . . . . .	138
7.12 VLAN usage . . . . .	140
7.13 Base Linux LAN notes . . . . .	140
7.14 NFS and SMB . . . . .	140
7.15 Conflicts with z/VM . . . . .	140
7.16 Secure (TLS) connection to z/OS . . . . .	141
<b>Chapter 8. zPDT licenses . . . . .</b>	<b>147</b>
8.1 Basic Concepts . . . . .	147

8.1.1	Types of tokens and licenses	148
8.2	Using a local zPDT system	149
8.3	UIM usage details	150
8.4	General zPDT client and server details	151
8.5	Client Installation and configuration for remote servers	152
8.5.1	Gen1 client configuration	152
8.5.2	Gen2 client configuration	153
8.5.3	Client UIM configuration	154
8.6	Server installation and configuration	155
8.6.1	UIM server	155
8.6.2	Gen1 License server	156
8.6.3	Gen2 License server	156
8.7	General Notes	158
8.7.1	Firewalls	159
8.7.2	Disk and Linux changes	159
8.7.3	Backup servers	159
8.7.4	Cloning zPDT	159
8.7.5	Removing functions	159
8.7.6	License expiration notification	160
8.8	Scenarios	160
8.8.1	Display serial number assignments	162
8.8.2	Security	163
8.8.3	Resetting UIM	164
8.8.4	SafeNet module restarts	164
8.8.5	Gen2 servers	165
8.9	Server search	165
8.10	Numbers	166
8.11	Dedicated key server	166
8.12	Gen1 token activation and renewal	167
8.12.1	Overview of Gen1 token updates	167
8.12.2	Gen1 token license update details (1090 tokens)	168
8.13	Summary of relevant zPDT commands and files	169
8.14	License manager glossary	170
<b>Chapter 9.</b>	<b>Other System z Operating Systems</b>	<b>173</b>
9.1	z/VSE	173
9.2	Linux for z Systems	173
9.3	z/VM	174
9.4	Installing the AD-CD z/VM 6.4 system	174
9.4.1	zPDT devmap	175
9.4.2	zPDT sensitivity	175
9.5	IPL and logon	175
9.6	Quick z/VM review	178
9.6.1	CMS	178
9.6.2	User MAINT	178
9.6.3	Minidisks and files	179
9.6.4	Inspecting your disks	180
9.6.5	XEDIT	182
9.6.6	z/VM directory	183
9.6.7	Spool contents	185
9.6.8	Simple system queries	186
9.6.9	zIIPs and zAAPs	187
9.6.10	Paging	187

9.6.11 Simple network adjustments . . . . .	188
9.7 z/TPF . . . . .	189
<b>Chapter 10. Multiple instances and guests.</b> . . . .	197
10.1 Multiple instances or guests . . . . .	197
10.2 Multiple guests in one instance . . . . .	198
10.3 Independent instances . . . . .	198
10.4 Instances with shared I/O . . . . .	201
10.5 Additional shared functions . . . . .	204
<b>Chapter 11. The awscmd command</b> . . . . .	207
11.1 Sample z/VM script . . . . .	208
11.2 z/OS use . . . . .	209
11.2.1 Sample z/OS program for awscmd . . . . .	210
<b>Chapter 12. Minor z/OS notes</b> . . . . .	217
12.1 How much storage for z/OS? . . . . .	217
12.2 DEFLATE and zEDC emulation . . . . .	217
12.2.1 zEDC adapter emulation in earlier zPDT releases . . . . .	218
12.3 Maintenance for AD-CD z/OS systems . . . . .	219
12.4 Java performance . . . . .	220
12.5 Excessive Health Checker messages . . . . .	221
12.6 z/OS spin loop timeouts . . . . .	222
12.7 PER traps . . . . .	222
12.8 Larger 3270 display . . . . .	222
12.9 z/OS disk STORAGE volumes . . . . .	223
12.10 Stand-alone z/OS dump . . . . .	224
12.10.1 Generating a stand-alone dump program . . . . .	224
12.10.2 Stand-alone dump output dataset . . . . .	225
12.10.3 Operating a stand-alone z/OS dump . . . . .	226
12.11 Moving 3390 volumes . . . . .	226
12.11.1 Create a source dump . . . . .	228
12.11.2 Send dump to Linux . . . . .	229
12.11.3 Receive dump . . . . .	229
12.12 IODF Changes with zPDT . . . . .	230
12.13 Many COBOL Compilations . . . . .	232
12.14 Local printing . . . . .	233
12.14.1 Setup . . . . .	234
12.14.2 Operational technique . . . . .	236
12.15 SYS1.LOGREC full . . . . .	236
12.16 Lost MVS console . . . . .	237
12.17 Unable to start ISPF . . . . .	238
12.18 Customized Offering Driver (COD) . . . . .	238
12.18.1 TCP/IP connection . . . . .	241
12.19 WLM and AD-CD . . . . .	243
12.20 TELNET . . . . .	245
12.21 Delete logstreams . . . . .	246
12.22 SMF . . . . .	246
12.23 Disabled waits . . . . .	247
<b>Chapter 13. Additional zPDT notes</b> . . . . .	253
13.1 Known z/VM issue . . . . .	253
13.2 Free zIIPs . . . . .	253
13.3 PC Hyper-Threading . . . . .	254

13.4	cpuopt statement	255
13.5	Copying DASD volumes	255
13.6	Read-only and shared DASD	256
13.6.1	Shared read-only volumes	256
13.7	zPDT log files	258
13.8	Very large PC memory	259
13.9	Token dates and times	260
13.10	Typing OPRMSG too many times	260
13.11	Important Linux command window	260
13.12	Linux “out of memory”	261
13.13	The crontab and sudo entries	261
13.14	Dynamic configuration changes	261
13.15	Security exposures	262
13.15.1	Reducing root usage	262
13.15.2	Linux suid use	263
13.15.3	Gen1 token server monitoring	263
13.16	z1090instcheck	263
13.17	zPDT build information	265
13.18	CKD versioning	265
13.19	zPDT messages	266
13.20	TCP/UDP ports	267
13.21	Remote operation	267
13.22	Many zPDT devices	267
13.23	Startup scripts	268
13.24	Suspend and Hibernation	269
13.25	Channel connections	270
13.26	Channel measurement blocks	271
13.27	x3270 scripting	272
13.28	Scenarios for awstape	272
13.28.1	Create a new labelled tape volume	273
13.28.2	Quick SL volume creation	274
13.28.3	Use an existing labelled tape	274
13.28.4	Automatically premount existing labelled tape	274
13.28.5	Use special CCWs to manipulate tape volume	275
13.28.6	Premounted tape	277
13.28.7	awstape utilities	279
<b>Chapter 14.</b>	<b>Tape drives and tapes</b>	<b>281</b>
14.1	zPDT 359x Tape Support	281
14.1.1	The FCP Adapters	281
14.1.2	3590/3592 Tape drives	282
14.2	Discontinued support	282
14.2.1	The awsscsi device manager	283
14.2.2	Parallel SCSI adapters	286
14.2.3	Specific hardware tested	286
14.2.4	zPDT SCSI utilities	288
14.2.5	Linux SCSI tape utilities	288
14.2.6	Practical advice	288
<b>Chapter 15.</b>	<b>DASD volume migration</b>	<b>289</b>
15.1	Warnings	290
15.2	Operational characteristics of the migration utility	291
15.3	Installation of the migration utility for z/OS	291

15.3.1	Server installation .....	292
15.3.2	RACF requirements .....	293
15.4	Operation of the server under z/OS .....	294
15.5	Installation of the server under z/VM .....	294
15.6	Operation of server under z/VM .....	295
15.7	The client commands .....	295
15.8	Additional notes .....	296
<b>Chapter 16.</b>	<b>Channel-to-channel .....</b>	<b>299</b>
16.1	z/OS use example .....	301
16.2	Multiple instances and z/VM .....	303
16.2.1	Devmaps .....	303
<b>Chapter 17.</b>	<b>Cryptographic usage .....</b>	<b>307</b>
17.1	Background information .....	307
17.2	Devmap specification .....	308
17.3	Initial ICSF startup .....	308
17.4	Operational notes .....	313
17.4.1	Multiple zPDT instances .....	313
17.4.2	Coprocessor control commands .....	314
17.4.3	New z/OS releases .....	315
17.4.4	Programming with CSF .....	315
17.5	Very basic dataset encryption example .....	317
17.5.1	z/VM usage .....	323
<b>Chapter 18.</b>	<b>Virtualization and containers .....</b>	<b>325</b>
18.1	Docker .....	325
18.2	Virtual environments .....	325
<b>Chapter 19.</b>	<b>Problem handling .....</b>	<b>329</b>
19.1	Problems starting zPDT operation .....	329
19.2	Problems during zPDT operation .....	331
19.3	Core images .....	333
19.4	Logs .....	333
19.5	Emulated volume problems .....	333
19.6	Special problem-related commands .....	334
19.7	Linux monitoring .....	335
<b>Chapter 20.</b>	<b>Server Time Protocol (STP) .....</b>	<b>337</b>
20.1	CCT uses .....	339
20.2	Configuration .....	339
20.3	Additional details .....	341
20.3.1	Leap seconds .....	342
<b>Appendix A.</b>	<b>FAQ .....</b>	<b>345</b>
<b>Appendix B.</b>	<b>Non-QDIO OSA .....</b>	<b>357</b>
<b>Related publications</b>	<b>.....</b>	<b>359</b>
IBM Redbooks	.....	359
Other References	.....	359
Help from IBM	.....	359
<b>Index</b>	<b>.....</b>	<b>361</b>

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®	PartnerWorld®	WebSphere®
DB2®	Passport Advantage®	z Systems®
Db2®	RACF®	z/Architecture®
DCP®	Rational®	z/OS®
FICON®	Redbooks®	z/VM®
IBM®	Redbooks (logo)  ®	z/VSE®
IBM z13®	Resource Link®	z13®
IBM z14®	S/390®	z15™
IBM z15™	System z®	zPDT®
MVS™	VTAM®	
Parallel Sysplex®	Watson™	

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Fedora, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VMware, and the VMware logo are registered trademarks or trademarks of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

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

# Preface

This IBM® Redbooks® publication provides both introductory information and technical details for the IBM System z® Personal Development Tool (IBM zPDT®), which produces a small System z environment suitable for application development. zPDT is a PC Linux application. When zPDT is installed (on Linux), normal System z Operating Systems (such as IBM z/OS®) may be run on it. zPDT provides the basic System z architecture and provides emulated IBM 3390 disk drives, 3270 interfaces, OSA interfaces, and so forth.

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

This book provides the primary documentation for zPDT and corresponds to zPDT V1 R10, commonly known as GA10.

## Authors

This book was produced in the International Technical Support Center (ITSO), Poughkeepsie. The author was Bill Ogden:

**Bill Ogden** is a retired Senior Technical Staff Member who continues to work part time with projects he enjoys. These include working with new mainframe users and entry-level systems.

The following people contributed substantially to this book:

**Keith VanBenschoten**, IBM Poughkeepsie, is the technical leader who provides zPDT test systems and the zPDT installation processes and tools.

**Theodore Bohizic**, IBM Poughkeepsie, is a Senior Technical Staff Member responsible for much of the design and implementation of zPDT.

**Frank Kyne**, with Watson™ and Walker, provides much-needed Sysplex assistance.

**Alena Yampolskaya**, IBM Poughkeepsie, is a member of the zPDT development team and has been very helpful in resolving questions about new commands and functions.

**George Darling**, IBM Poughkeepsie, contributes updates and understand of I/O elements and specific z System architecture.

## | 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 ITSO residency project and help write a book 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](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- Mail your comments to:

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

## Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<https://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

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



# Introduction

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

This document is not intended as an introduction to IBM System z or to z/OS. Readers are assumed to have background knowledge in these areas, and terms common to these areas are used throughout this document without additional introduction. This document corresponds to Version 1 Release 10 (“GA10”) of zPDT. GA10 provides IBM z15™ architecture. Be aware that any operating systems used with this zPDT release must be capable of IPLing and running on a z15. Some older operating systems might require service in order to run on a z15

## Important:

See 2.8.1, “Current release (GA10)” on page 24 for specific information about z/OS operation with compressed data.

## 1.1 General architecture

IBM has long encouraged the use of several small IBM S/390®<sup>1</sup> environments for use by the IBM development community<sup>2</sup>, and these have proven useful. zPDT provides several functions that extend the usefulness of small System z development systems. These include System z instruction compatibility<sup>3</sup>, speciality processors,<sup>4</sup> cryptographic adapter functions,

<sup>1</sup> The reference to S/390 is for the historical context of this paragraph.

<sup>2</sup> In this context, we primarily refer to the IBM PartnerWorld® for Developers organization (previously known as Partners in Development).

<sup>3</sup> Several instructions that are not relevant for zPDT are not provided.

<sup>4</sup> These are IBM System z Integrated Information Processors (zIIPs), IBM System z Application Assist Processors (zAAPs), and IBM Integrated Facility for Linux Processors (IFLs).

channel-to-channel operations, system timer protocol functions, and coupling facility functions.

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

The basic zPDT function consists of the zPDT software (processor functions, device emulators, utilities) plus a “license” needed to execute the zPDT functions. This license is provided by a local or remote hardware USB key device (“token”) or by a software-only license server.<sup>5</sup> The license must be accessible when zPDT is being used, but can be removed at other times. Chapter 8., “zPDT licenses” on page 147 goes into more detail about license and token types.

In most cases, the underlying Linux PC (that is used to install and run the zPDT system) should have *at least* one more PC core than the number of zPDT CPs used in the largest zPDT instance.<sup>6</sup> The base Linux machine that is used for zPDT must have sufficient memory. No specific size is required, but 8 GB should be regarded as a bare minimum and most zPDT systems have considerably more memory than this.<sup>7</sup> Disk space is needed for emulated 3390 (or 3380 or FBA) volumes and a typical zPDT base machine will have *at least* 200 GB of disk space.

zPDT systems can have up to eight emulated System z CPs<sup>8</sup>, limited by the number of zPDT licenses available. The System z architecture levels for the CPs are indicated in Table 1-1.

Table 1-1 System z architecture levels

Release date zPDT	zPDT release	zPDT build level	System z architecture	ARCH level (for compilers)
2009, 2010	V1R1 “GA1”	39.xx	z800, 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)
2015	V1R6 “GA6”	49.xx	IBM z13®	ARCH(11)
1Q 2017	V1R7 “GA7”	49.xx	IBM 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)

<sup>5</sup> License details (including tokens) are described in “zPDT licenses” on page 147.

<sup>6</sup> zPDT *requires* that the number of PC cores available must at least equal the number of CPs (and zIIPs, zAAPs, IFLs) configured in the devmap. If fewer cores are available, zPDT automatically reduces the number of emulated System z processors.

<sup>7</sup> For users intending to install z/OS 2.3 (or later) we suggest 16 GB as a minimum PC memory size.

<sup>8</sup> This maximum count includes general purpose CPs, zIIPs, zAAPs, and IFLs.

This document provides the primary documentation for zPDT Version 1Release 10 (“GA10”). Material about sysplex operation that was in prior editions has been moved to a new document, *zPDT Sysplex Extensions*, SG24-8386.

## 1.2 zPDT security, integrity, and RAS concepts

zPDT emulates System z architecture while running as a Linux application. zPDT has no control over the security or integrity environment of this “base” Linux. While we believe that zPDT generally follows reasonable Linux application standards, zPDT should not be considered a secure system unless all aspects of access to the base Linux are also considered. Within zPDT itself (while running an IBM operating system) the normal security and integrity of that environment exists.<sup>9</sup> For example, the z/OS AD-CD package typically available to zPDT users contains IBM RACF®, and this 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 stored in these emulated devices. A malicious user could “front end” various zPDT administrative commands (which run as ordinary Linux commands), although zPDT provides some protection against this exposure. A small number of zPDT modules run with SUID to *root*; a malicious user with access to these modules could modify them.

Ideally, access to the base Linux system running zPDT could be limited to only the necessary trusted administrative personnel. General user access to the System z operating system running under zPDT would be only through System z interfaces, such as emulated 3270 terminals.<sup>10</sup> Such a restrictive environment is not always possible and, even where intended, skills are needed to create and maintain the environment.

*zPDT does not provide the reliability, scalability, or serviceability (RAS) of a standard System z.* zPDT has no control over exposures inherent 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 System z. Prudent users should have well-defined backup procedures (for emulated volumes), procedures for monitoring Linux and z/OS consoles (for error messages), periodic checking of emulated volume structure validity (using the `a1cckd -rs` function), and monitoring to verify that the system is not routinely overloaded.

**Important:** zPDT is intended for development work. It is not intended as a secure system. We suggest you think twice before moving any confidential data to your zPDT systems.

<sup>9</sup> A notable exception is that cryptographic keys for emulated cryptographic adapters are stored in standard Linux files. These are secure from the System z viewpoint, but not from the Linux viewpoint.

<sup>10</sup> 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 zPDT customers, but might provide a security exposure for other customers.

**Important:** zPDT shares its host Linux system with whatever else is running in that Linux system and has no control over resources used or exposures created by other Linux applications. Operating systems running under zPDT (such as z/OS) have internal timing dependencies and might have problems if starved of resources. zPDT, with z/OS for example, can be a major load for a smaller PC. Common sense is needed for managing configurations, but we cannot specify a formula for reasonable or maximum system loading. zPDT, running as the only significant application, with at least one more PC core than zPDT CPs (including zIIPs and so forth), and with sufficient PC memory should not experience problems with reasonable workloads. Significant other Linux applications, such as Linux IBM WebSphere®, as an example, running on the same small PC with zPDT can produce an unreasonable system load that encounters errors.

This issue of “common sense” becomes complex in a virtualized environment where overcommitment can create error situations that are difficult to detect and debug.

## 1.3 Terminology

The first releases of zPDT were for Independent Software Vendors (ISVs) who were qualified through IBM partner programs. The IBM offering name and the program package name was simply “zPDT” and it required a license token of IBM type 1090. The terms “1090” and “zPDT” were used interchangeably with no confusion.

The zPDT “program” is a package with many programs inside it, although we often refer to it as a single “program.” The multiple internal programs include the emulator program itself, various device emulation programs, and many smaller Linux programs that constitute the Linux commands to manage zPDT.

A later offering was under the name IBM Rational® Development for z Unit Test (RDzUT), subsequently changed to Rational Development and Test (RD&T), and again changed to zD&T. This version required a license token type 1091. As the following table indicates, the base *program* in these offerings is zPDT, while the current *product names* are zPDT or zD&T.

**Table 1: Product offerings and license types**

Offering “Product”	Base Pgm	Gen1 token model	Gen2 license number	emulated CP model	Prior names
[ISV] zPDT	zPDT	1090	not used yet	1090	z1090 (original), zPDT
zD&T	zPDT	1091	333	1090	RDzUT, RD&T (an option includes CF functions)
AD-CD	zPDT	1090 or 1091	334	N/A	AD-CD “decryption license” for ISV zPDT and zD&T

The license control mechanisms are from Gemalto N.V., under the general product name of SafeNet. The 1090 and 1091 tokens are from an older SafeNet product family we refer to as Gen1. We refer to a newer SafeNet family of license control mechanisms as Gen2. The Gen2 licenses can be through a software-only control (no token) or through a new family of hardware tokens.<sup>11</sup> The zPDT program always identifies the emulated System z type as 1090, regardless of the token type or Gen2 license number.

<sup>11</sup> The new Gen2 hardware tokens may become available for zPDT in the future. Support for existing Gen1 tokens (1090 and 1091) will continue.

During zPDT development the Gen1 tokens were sometimes referred to as SHK tokens or licenses, and the Gen2 tokens (and software-only license server) were referred to as LDK licenses. You might see these references in some documentation.

The zPDT program has minor differences between the ISV version and the zD&T version. For example, the ISV version will not function with a 1091 token and vice versa. The zD&T version requires an additional license feature in order to run z/VM or Coupling Facility code. Other than differences at this level, the technical descriptions in this document apply to both versions. In the few cases where differences exist, the text should make this apparent.

The installation processes for the ISV and zD&T versions can be quite different,<sup>12</sup> but this is external to the operation of the zPDT program, once installed. This document describes the basic installation process for either version. The administrative processes to obtain license updates differ between the ISV and zD&T offerings. In both cases, the user must work with his zPDT provider to obtain license updates.

In summary, “zPDT” is typically used to refer to the basic emulation program, but it can also be a more restrictive reference to the ISV offering to differentiate it from the zD&T offering. A much more extensive discussion of tokens and licenses is in Chapter 8., “zPDT licenses” on page 147.

## 1.4 PC choices

zPDT (whether ISV or zD&T version) is generally not sensitive to the brand or model of the underlying PC. We see systems ranging from rather old laptops (2 cores, 8GB memory) to very large servers (24 cores or more, up to 256 GB memory). Considerations for selecting a PC include the following:

- ▶ Is absolute peak performance required? The fastest machine we have seen (at the time of writing), in terms of raw emulated CPU performance, is a large server,<sup>13</sup> with second place to a high-end laptop. *However*, the performance differences among recent PCs (assuming they are reasonably configured for the workload involved) are not very large.
- ▶ How important is RAS? Elements include RAID, dual power supplies, memory recovery technology, inclusion of support processor functions, battery backup, and so forth.
- ▶ Do you need a graphic desktop? Laptops are good at this, but servers are often rack mounted in an “operations” area, without a good display, keyboard, or mouse.
- ▶ How much memory do you need? This is a complex topic. In this book we recommend *at least* one gigabyte more than the emulated System z size, but this is quite minimal. The Linux disk cache is a very important performance element and it depends of sufficient memory. Likewise, the ability to avoid Linux swapping is important for performance. As a starting point, the author suggests *at least* 8GB above the zPDT memory defined in the devmap. Typical zPDT systems might have PC memory size at least twice as large as the defined zPDT System z memory.
- ▶ How many cores do you need? zPDT requires a number of cores at least equal to the number of zPDT CPs and zIIPs. Again this is minimal, where two or three additional cores can improve performance. If significant additional workloads are present in Linux, then the number of cores needed is likely to be larger.
- ▶ Do you plan a virtual environment? This always has some performance implications, but can have a major impact if the server is over committed.

<sup>12</sup> The most basic installation processes, described in this document, are the same. However, the zD&T package might provide “front ends” intended to simplify installation. Likewise, your zD&T or ISV zPDT vendor might provide a different “fron end” installation process.

<sup>13</sup> See 8.11, “Dedicated key server” on page 166 for details.

- Does your z/OS system (including expected applications and subsystems) need large memory? A z/OS system with a number of TSO users editing and submitting compilations might be happy (that is, not paging) with 8GB, whereas a large IBM Db2® workload using lots of buffers might be happier with 100+GB. We have observed a tendency to overestimate the z/OS memory needed (in the devmap), and underestimate the importance of memory for Linux. Of course, available memory is not dedicated to either zPDT or the disk cache but is managed by Linux as best it can.
- Do you have a “real” Internet IP address, as opposed to a DHCP address? If so, these tend to not be very portable, being defined to a specified subnet. In turn, this implies a fixed-location server as opposed to a portable laptop.

## 1.5 Installation and operation

This document describes basic methods of installation and operation of zPDT. There are extensions not described in this document. For example, both zD&T and ITC<sup>14</sup> provide graphic interfaces that can be used to install and operate zPDT. These “higher level” interfaces are described by their own (zD&T or ITC) documentation. However, it should be possible to revert to the basic installation and operation commands described in this document if necessary.

---

<sup>14</sup> Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 or [sales@itconline.com](mailto:sales@itconline.com).

## Function, releases, content

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

- ▶ The *base machine*, or *underlying host*, or *underlying Linux*, or *host Linux* is the Intel-compatible personal computer (PC) that is running Linux.
- ▶ z/OS is used to refer to any recent release of the z/OS operating system. Likewise for z/VM, and so forth.
- ▶ A device map, or *devmap*, is used to specify the operational configuration of zPDT. It is a simple Linux flat file.
- ▶ A *token* or [license] *key* refers to a hardware device that supplies a zPDT *license*. The *token*, *key*, and *license* terminology is sometimes used interchangeably. A *license* can also be supplied by a software-only mechanism.
- ▶ *Processor* or *core* normally refers to the Intel or AMD processor cores in the base machine. A two-core machine has two processors in this terminology, although both are typically in one hardware “processor” module. There is always some confusion in this terminology.
- ▶ *CP* refers to a general System z processor and is the major functional element of zPDT. By default, zPDT provides System z CPs. You may optionally convert a CP to a zIIP, zAAP, or IFL processor.<sup>1</sup>
- ▶ *Open Systems Adapter (OSA)* is sometimes used as shorthand for an *OSA-Express adapter*. The zPDT GA10 system currently provides OSA-Express5s emulation.
- ▶ Many Linux commands, for the *base Linux* system, are shown throughout this book. 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 running completely as *root*.
- ▶ We mention zPDT releases as GA6, GA7, GA8 and so forth. This is an abbreviated terminology. GA10, for example, means zPDT Version 1 Release 10. All zPDT releases, to date, have been Version 1. The “GA” abbreviation is common terminology from IBM

<sup>1</sup> Using more general System z terminology, zPDT provides PUs. By default, the PUs are characterized as CPs, but may be characterized as zIIPs, zAAPs, or IFLs instead. Throughout this document we generally refer only to CPs and this reference should be understood to include zIIPs, zAAPs, and IFLs when these are used. zAAPs are no longer available at the z14 level, but were available in earlier zPDT releases.

meaning “General Availability.” The term GA10.1 means the first “fixpack” for the GA10 release, and so forth.

- ▶ System z I/O devices have *device numbers* used to specify a specific device or interface. An older terminology for a device number is *address*, and this older terminology is still widely used although it is potentially confusing in certain cases. This document uses *address* and *device number* interchangeably and the author apologizes to anyone who is upset by this usage.

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

## 2.1 System z characteristics

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

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

System z software running in a zPDT environment is binary compatible with large IBM System z machines. For example, application programs compiled and linked in one environment can run unchanged in the other environment,<sup>2</sup> assuming that configuration elements are compatible.

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

Not all System z instructions and functions are available with zPDT. Instructions 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 IPL and Internal IPL
- ▶ The accelerator, PKCS#11, and customized cryptographic routines (UDX) functions of cryptographic coprocessors.
- ▶ Time of Day (TOD) steering
- ▶ IBM enterprise Blade Center Extension (zBX) functions
- ▶ CPU Measurement Facility
- ▶ Asynchronous data movers
- ▶ IBM FICON®, and Transport Mode I/O

<sup>2</sup> This general statement assumes that relevant operating system and other libraries are at compatible levels, and so forth. It also assumes that relevant licenses allow such use.

- ▶ Parallel Access Volumes (PAV)
- ▶ Logical channel subsystems
- ▶ IBM Hiper Sockets functions
- ▶ LPARs
- ▶ Flash memory
- ▶ Multiple I/O paths
- ▶ Multithreading (MT or symmetric MT (SMT))
- ▶ Various fields within the Channel Measurement Facility reports
- ▶ Hardware Management Consoles (HMCs) are not available.
- ▶ Some CHSC commands (such as crypto adapter measurements)
- ▶ System Recovery Boost
- ▶ The DFLTCC instruction.
- ▶ MSA9 functions (in zPDT GA10)

IBM z15 systems do not have zAAP speciality processors. zAAP speciality processors are mentioned throughout this book for compatibility with earlier zPDT releases.

### 2.1.1 Architecture levels

IBM System z machines have architectural characteristics, such as new instructions on newer systems or changed firmware characteristics. The architectures are not always completely “backward compatible” and may require software updates to run older operating systems on newer architectures. For example, z15 machines (including zPDT GA10) will not run with the original z/VM 6.2 release. In this case, a PTF is needed to resolve the incompatibility. Such software updates are often known as “toleration PTFs” and are usually available for a certain number of older operating system releases when a new System z series becomes available. IBM does not provide such updates for much earlier operating systems.

**Important:** Operating systems run on zPDT GA10 must be configured for use with IBM z15 machines. There are limitations about IPLing older operating systems.

## 2.2 Hardware token

A zPDT system is not functional without a machine-readable zPDT license. This can be from a local USB token (a Gen1 1090 or 1091 token, or a newer Gen2 token<sup>3</sup>), or from a remote license server (with 1090, 1091, Gen2 token, or Gen2 software-only licenses). The typical USB tokens (“hardware keys”) are shown in Figure 2-1. A 1090 key is at the top of the illustration and should always have a tag attached to it. A 1091 key is at the bottom of the illustration and usually has a blue or green color code. The serial number of the 1090 key is printed on the tag. The serial number of the 1091 key is engraved on the back of the key.

Throughout this document we frequently mention tokens. However, the equivalent zPDT (or zD&T) license control is available through remote license servers where tokens (or special “software-only” licenses) are present. When our text mentions “tokens” or “keys” you should be aware that this includes the potential use of these remote license functions.

<sup>3</sup> These are not yet available at the time of writing.



Figure 2-1 The 1090 and 1091 hardware keys

If the token is removed while zPDT is operational or if the connection to a remote license server is lost, the operation pauses with a series of messages, ending with these:

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

If the intervening time interval does not disrupt the operating system or application programs, zPDT operation can be resumed by connecting the license again.

A USB token (or other license form) is normally valid for one year after it is initialized or *activated*. It can be reinitialized<sup>4</sup> at any time, which normally extends the validity for a year beyond the date of the most recent reinitialization.<sup>5</sup> The procedure for initializing the key (or reinitializing it) depends on the channel you used to obtain your zPDT system. This might be through an IBM Business Partner or some other supplier. See “Gen1 token activation and renewal” on page 167 for more about token updating.

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

Starting with zPDT GA8, zIIPs do not “count” when considering the number of licenses in your token(s). This is discussed further in 13.2, “Free zIIPs” on page 253.

Various “SMP effects” reduce the effectiveness of additional CPs. For example, going from seven to eight CPs might offer minimal performance enhancements for many workloads. The

<sup>4</sup> This is also known as a “lease extension.”

<sup>5</sup> The extension period might differ depending on the IBM channel used to obtain the zPDT system.

I/O capability of the underlying PC must also be considered. However, this performance determination is left to you.

## 2.2.1 Emulated I/O

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

<b>aws3274</b>	Emulates a local, channel-attached 3274 control unit. This device manager is almost always used to provide the IBM MVS™ console, for example, and 3270 application sessions. Each terminal appears (to the System z operating system) as operating through a channel-attached non-SNA DFT IBM 3274 control unit. TN3270 sessions are used, via the base Linux TCP/IP interface.
<b>awsckd</b>	Emulates 3390 (and 3380) disk units, using a single Linux file for each 3390 or 3380 device.
<b>awsosa</b>	Emulates an OSA-Express5s adapter, in either QDIO (OSD) or non-QDIO (OSE) mode. The hardware involved is an Ethernet adapter on the underlying PC. <sup>6</sup> This device manager can support TCP/IP operation. SNA operation is not supported. <sup>7</sup> It can also support OSA/SF usage where appropriate.
<b>awstape</b>	Emulates a 3420, 3480, 3490, or 3590 tape drive, using a Linux file in place of the tape media.
<b>awscmd</b>	Emulates a tape drive but routes output records to the base Linux system where they are executed as commands and returns Linux output to the emulated tape drive.
<b>awsfba</b>	Emulates FBA devices, which are supported by IBM z/VSE® and z/VM. A Linux file is used for each emulated device. Note that this is not the Fibre Channel (Open Systems) FBA on recent System z machines.
<b>awsoma</b>	Emulates the Optical Media Attach interface, working with Linux files in this format. This function is read-only.
<b>aws3215</b>	Emulates a 3215 console device (seldom used today), using a Linux terminal window for the interface.
<b>awsprt</b>	Emulates a 1403 or 3211 printer, using a Linux file for output. Provides emulation of a 1403-N1 or 3211 printer. FCB emulation for 3211 is provided, but UCS functions are not provided. Automatic ASCII translation (fixed translation table) is provided.
<b>awsrdr</b>	Emulates a 2540 card reader, using Linux files as input. (The 2540 card punch functions are not emulated.) Both EBCDIC and ASCII data may be used.
<b>awsscsi</b>	Uses a Linux SCSI-attached tape drive as a System z tape drive, providing a way to read/write “real” mainframe tape volumes. See Chapter 14, “Tape drives and tapes” on page 281 for specific drive and adapter details.
<b>awsctc</b>	Emulates an IBM 3088 channel-to-channel adapter by using TCP/IP as the communication mechanism. The connection can be the same zPDT instance, another instance in the same PC, or a zPDT instance in a LAN-connected machine.

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

<sup>6</sup> Wireless can be considered an Ethernet adapter.

<sup>7</sup> Initiating SNA operations (in non-QDIO mode) might be possible, but this usage has not been tested and is not supported by IBM at this time.

The emulated I/O support is summarized in Table 2-1 on page 12.

Table 2-1 Emulated I/O summary

Manager	Control unit	Emulated device	Model
aws3274	3274	3279 (and rarely, 3284)	various
awsrdr	2821	2540 card reader	
awsprt	2821, 3811	1403, 3211 printers	
awsckd	3990	3380, 3390	1, 2, 3, 9 <sup>a</sup>
awstape	3803, 3480, 3490	3420, 3422, 3480, 3490, 3490E, 3590	
awsfba	3990	9336	1, 2 <sup>b</sup>
awsoma	3803	3422	OMA
awsscsi	3480, 3490, 3590	3420, 3480, 3490, 3592	
awsosa	OSA	OSA	
aws3215	3215	3215	
awsctc	3088	3088	

a. Model 9 refers to 3390s. Actually, a 3390 with any valid number of cylinders may be defined and used, including EAV units.

b. The model emulated depends on the number of blocks defined, although z/VSE can force a model selection.

The current zPDT design allows a maximum of 2048 emulated I/O devices; these are often described as 2048 *subchannels*. A practical number of emulated I/O devices might be considerably less than 2048, depending on many factors.

## 2.2.2 Concurrent PC workloads

An emulated System z processor, especially when running z/OS, must provide sufficient processing power to meet basic requirements. z/OS has various timers running to detect error situations. Sufficient processing power (for *each* CP, if multiple System z CPs are used) must be available to prevent these timers from expiring. Insufficient processing power can result in SPINLOOPS, MIH<sup>8</sup> actions, OSA communication drops, or other apparent I/O device error problems.

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

Reasonable care is needed even when extra base processor cores are available. For example, performing large Linux disk copies<sup>9</sup> while the System z function is operational can effectively lock out normal System z work and create time-out situations.

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

<sup>8</sup> Missing-interrupt handler

<sup>9</sup> This can include using **gunzip** on AD-CD z/OS volumes while zPDT is active.

## 2.3 Operational overview

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

### 2.3.1 Linux userids

In principle, any Linux userid may be used to install<sup>10</sup> or operate zPDT, with the exception that a zPDT operational Linux userid cannot be longer than eight characters (unless the *system\_name* statement is used in the devmap, in which case there is no special limit on the Linux userid length). All our examples assume userid *ibmsys1* is used, but there is nothing special about this name. The zPDT system uses several default path names that are related to the current Linux userid.

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

Current Linux operating systems automatically create home directories for userids in the format */home/<userid>*. For example, the home directory for userid *ibmsys1* is */home/ibmsys1*. It is possible to specify a different home directory for a userid. Throughout this document we often use */home/ibmsys1* to indicate the home directory for the zPDT userid, even though the specific use of *ibmsys1* is not required.

### 2.3.2 zPDT instances

Logging into Linux and starting a zPDT operation creates an *instance* of zPDT usage. This instance might have one or more System z CPs associated with it, depending on the licenses available and the parameters in the devmap. If you then log in to Linux with a second Linux userid, and start another zPDT operation, this creates a second *instance*. Multiple instances means that multiple, independent zPDT environments are run in parallel. The total number of CPs across all concurrent instances cannot exceed the number allowed by the token.<sup>11</sup>

A Gen1 1090 model L03 token can have up to three System z CPs (or mixtures of CPs, zAAPs, and IFLs) plus up to three zIIPs. These could be used for three zPDT instances, each with a single CP and separate System z memory<sup>12</sup> and a separate System z operating system. Alternatively, a single zPDT instance could be used with one, two, or three CPs; this is the more likely usage for most zPDT users. The use of multiple CPs is subject to the following restrictions and considerations:

- ▶ The number of defined CPs (including zIIPs, zAAPs, or IFLs) in one zPDT instance *should* be *at least* one less than the number of processor cores on the base Linux system. If there are fewer cores than requested CPs, zPDT does not start all the requested CPs.
- ▶ Full zPDT operation can use more cores in the base PC system than there are System z CPs defined in any one instance. The additional processors are used for I/O, to help prepare System z instructions for use, and for non-zPDT Linux processes.

It is possible to use more than one token. For example, a machine with two model L03 tokens would have a maximum of six CPs. A significant “multiprocessor effect” is present and the advantages of more than four or five CPs might be marginal, depending on the nature of the

<sup>10</sup> Part of the installation process must be done as *root*, but the initial login should be with the userid that will be used to operate zPDT.

<sup>11</sup> Or by the total number of licenses from multiple tokens or software licenses, with a design limit of eight for any given zPDT instance.

<sup>12</sup> The combined System z memory is subject to the later discussion about memory.

workload. Also, the I/O limitations of the underlying PC become more relevant when using more than three CPs.

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

Most of this document is focused on single instance operation. “Multiple instances and guests” on page 197 provides setup and usage instructions for multiple zPDT instances.

### 2.3.3 zPDT console

A zPDT system is operated from Linux command lines.<sup>13</sup> This operation could be done remotely through Telnet or SSH connections. A graphics connection is not needed.

There is no dedicated console program for sending commands to an operational zPDT environment.<sup>14</sup> All zPDT commands are Linux executable files that are executed from a Linux shell prompt. The commands *require* that the zPDT instance be started by the same Linux userid that issues the subsequent zPDT commands for that instance. For example, if Linux userid `ibmsys1` starts zPDT then only Linux userid `ibmsys1` can issue an `ip1` command. The `ip1` command is a Linux executable file, supplied with the other executables that constitute the zPDT package.

zPDT sometimes issues asynchronous messages. These are sent to the Linux command window that was used to start that zPDT instance. If that command window is closed, the asynchronous messages are not seen.<sup>15</sup> You can issue zPDT commands from any Linux command window running under the Linux userid that started the zPDT instance.

Do not confuse the MVS operator console (for example), running in a 3270 window, with Linux terminal commands, which are run from a Linux command window.

### 2.3.4 Performance

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

- ▶ Performance depends on the power of the underlying hardware and this changes frequently. Performance is related not just to the clock speed of the underlying processor (such as 3.4 GHz for an Intel processor) but is also related to the memory design and the pipelining, caching, and translation design of the underlying processor.
- ▶ Linux performance (including applications such as zPDT) can be greatly influenced by how the Linux disk cache (and swap file) is performing, and the nature of the Linux disks.
- ▶ The number of CPs used by 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 zPDT can change performance.
- ▶ The System z instruction mix and memory reference pattern has a profound impact on performance—a greater impact than is observed on a larger System z.

<sup>13</sup> As mentioned earlier, “higher level” operator interfaces are provided by zD&T and ITC.

<sup>14</sup> Do not confuse zPDT commands with z/OS operator commands.

<sup>15</sup> However, the messages are added to a console log file. See “zPDT log files” on page 258 for more information about these log files.

- ▶ MIPS (million instructions per second) is a rather discredited metric, although it is still informally used with the smaller System z machines. Any MIPS number for 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 zPDT might be from a single (relatively slow) computer disk drive, or might be from solid-state disks. Workloads with modest I/O loads (when run on a larger System z) might be completely I/O-bound on a zPDT system.
- ▶ Virtualization can add much more variability, especially when the host computer is overcommitted.

As an example, zPDT on a Lenovo W530 notebook computer provides reasonable performance for running z/OS, with typical TSO and batch usage, small IBM Db2 usage, and so forth. Using emulated local 3270 connections, reasonable performance might be maintained for a number of such users. The general “look and feel” for such usage generally provides subsecond response typical of smaller System z installations.

The zPDT design goals are based on the assumption that it is the only significant application running on the host Linux machine. The impact of additional substantial applications is most significant for Linux memory management and cache management. This can be considerably more important than the extra CPU cycles taken by another application.

z/VM may be used with zPDT. The performance of guest operating systems under z/VM (such as z/OS running under z/VM) is influenced by the use of the SIE function. On a large System z, this function provides a “microcode assist”<sup>16</sup> for many of the virtualization functions performed by z/VM. Most SIE functions are provided by zPDT, but there is no direct equivalent of a “microcode assist” level and the virtualization performance boost provided by SIE is modest.

zPDT may be used in Linux shared file environments, such as provided by the Network File System (NFS). Depending on how this is implemented, it can produce substantial performance degradations. We suggest you test such use, in your operational environment, before designing a configuration that depends on shared file operation.

## Extreme configurations

IBM has not tested extreme zPDT configurations. For example, in theory a zPDT instance can have up to 2048 devices, up to 8 CPs, and a base Linux can have up to 15 concurrent zPDT instances. In a wildly extreme configuration, this might represent 15 \* 2048 or over 30,000 emulated devices being used by 120 CPs. Extreme configurations, considerably smaller than this example, might not be practical. Among other considerations, each emulated device requires control blocks in Linux shared memory and a very large configuration might cause difficulties with Linux shared memory and swap file configurations

Excessively busy storage devices (such as a large NAS device) that have prolonged response times because of heavy loading (such as copying large amounts of data while zPDT is attempting to use the device for emulated DASD) might cause problems.

zPDT is not intended to replace normal System z configurations. If you are considering a configuration with more than, say, a hundred emulated devices, or with very heavily loaded I/O devices, or with many z/VM guests, we suggest discussing your requirements with your zPDT supplier. Moderately large configurations *are* possible, but we suggest you review your plans with knowledgeable zPDT people. For such configurations you must determine the suitability of zPDT for your requirements. **The key to understanding zPDT capacity and performance is a *detailed* understanding of *your* workload.**

<sup>16</sup> This is the common terminology for SIE operations, although the actual implementation might be much more complex than implied by this statement.

## 2.4 Base configurations

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

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

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

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

The basic zPDT program does not include any System z software. Although System z software might be part of an offering that includes zPDT, the base zPDT product itself does not include any System z software. System z software must be obtained in a media format suitable for a zPDT machine.

### 2.4.1 Hardware and software levels

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

#### Base Linux

zPDT Version 1 Release 10 “GA10” was built for operation on the Linux levels listed in 2.8.1, “Current release (GA10)” on page 24. These are the “supported” base Linux releases. Earlier Linux distributions should not be used due to potential Linux library differences. Various Linux distributions 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:** A useful System z installation, even as small as a zPDT system, can represent a major investment for the owner. The zPDT development team assumes serious users will select one of the supported Linux bases that have been extensively tested with zPDT. Over time zPDT will follow general Linux developments and changes, but constantly chasing the latest Linux distributions is not a primary zPDT goal.

A suitable 3270 emulator is needed, but this is not distributed with zPDT. Many 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 zPDT must be verified by you. IBM developers have used recent releases of the IBM PCOMM package (on Microsoft Windows systems).

*Do not confuse* the following two Linux systems:

- ▶ The Linux you install on your PC in order to run zPDT. This is your *base Linux*.
- ▶ The *Linux for System z* that you might elect to run under zPDT.

These are completely separate topics. With few exceptions, all mentions of “Linux” in this book refer to the Linux you install on your PC.

## Base PC hardware

zPDT Version 1 Release 10 was tested on the PC hardware listed in 2.8.1, “Current release (GA10)” on page 24. The systems listed are the only *tested* machines for zPDT. Other machines *might* work correctly with zPDT, but they have not been tested. In rare cases, IBM might address zPDT problems only when reported on one of the tested machines.<sup>17</sup> The zPDT system has no specific requirements for these particular base machines and operating systems, but the almost infinite number of possible combinations of other hardware and other Linux versions have not been tested.

Additional notes on the hardware include the following:

- ▶ In all test cases, a minimum of 16 GB PC memory was available. Systems with up to 256 GB have been used.
- ▶ zPDT requires at least as many PC cores as there are CP/zIIP/IFL processors defined to zPDT. Performance is improved by having *at least* one more PC core than there are CP/zIIP/IFL processors defined to zPDT.
- ▶ Various USB disks were used to the extent supported by Linux.
- ▶ At the time of writing, the use of “bonded Ethernet interfaces” is not supported.
- ▶ At the time of writing, network-attached storage (NAS) disks have had mixed reviews by zPDT users. The issues are at the Linux level. 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 use of the NAS disks, we suggest that they are not appropriate for zPDT use. In some cases, the usability might be related to congestion and bandwidth of the LAN involved.
- ▶ A suitable USB port must be available for the hardware token.<sup>18</sup> Do *not* use an unpowered USB port expander when using zPDT. (The license server function, described in Chapter 8, “zPDT licenses” on page 147, provides an alternative way to manage licenses.)
- ▶ Multiple LAN interfaces might be useful in larger configurations, although this is rare.
- ▶ We suggest disabling the hyperthreading (if available) at the BIOS level. Hyperthreading can produce slowdowns when z/OS is running spinloops. If many PC cores are available the slowdowns might be resolved before z/OS console messages are produced, meaning there is no indication of a problem other than reduced performance. This is discussed further in 13.3, “PC Hyper-Threading” on page 254.
- ▶ 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 may be used with zPDT. They can be used through Linux utility functions or used directly by the System z operating system (where they appear to be IBM 3420, 3480, 3490, or 3590 tape drives). Not all SCSI tape drives are usable by zPDT. The usability depends on the exact model, the exact firmware level, the exact SCSI adapter used, and the firmware options that are set in the drive. IBM has used a variety of different SCSI drives for testing, but IBM cannot predict whether *your* SCSI drive will work with zPDT. If this is important to you, we *strongly suggest* that you discuss your requirements

<sup>17</sup> As of the time of writing, this situation was encountered only once and was due to attempted use of a very old PC.

<sup>18</sup> The use of a remote license server is possible. In this case a USB port is not needed.

with your zPDT provider. See Chapter 14, “Tape drives and tapes” on page 281 for more information.

## 2.5 Using older System z architectures

zPDT does not have a facility to emulate older System z architectures. For example, the current release (zPDT Version 1 Release 10) is at the z15 level. It cannot be set to a System z 196 level or a z10 level, for example. Providing a switchable architectural level facility would result in reduced performance and the zPDT developers are unwilling to make this tradeoff.

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

## 2.6 zPDT Components

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

- ▶ A base Linux system. This is not provided with zPDT. You must acquire this directly.
- ▶ A suitable 3270 emulator (which is usually run on the same personal computer that is hosting zPDT, although this is not required). The zPDT package does not provide a 3270 emulator.
- ▶ The hardware USB token<sup>19</sup>, which is required for zPDT operation. (An alternative is a license server; see Chapter 8., “zPDT licenses” on page 147 for a larger discussion of the options.)
- ▶ A zPDT program package file.<sup>20</sup> Within this single file are the following items:
  - Two prerequisite SafeNet driver programs for communicating with the 1090 or 1091 tokens. These two drivers are provided with zPDT and only these provided versions may be used. Other versions 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 will be used.
  - A program for communicating with Gen2 tokens or license servers.
  - The Red Hat (RHEL, Fedora) version of zPDT.
  - The Novell (SLES, openSUSE) version of zPDT.<sup>21</sup>
  - The Ubuntu version of zPDT.
  - An installer program that displays a license, installs the prerequisite drivers (if not already present), and then selects and installs the correct zPDT version.
  - Components that provide remote license and identity management functions.

System z software, such as z/OS, is not part of zPDT. It must be licensed and acquired separately.

<sup>19</sup> This is a Gen1 1090 or 1091 token, or possibly a future Gen2 token.

<sup>20</sup> There are two separate program file packages, one for 1090 (ISV) systems and one for 1091 (zD&T) systems.

<sup>21</sup> This is not included with the zD&T versions of zPDT.

The remainder of this section discusses the components in zPDT (after it is installed). The discussion is the same whether the Red Hat, Novell, or Ubuntu versions are used and whether the ISV or zD&T package is used. Different versions of zPDT (for Red Hat, Novell, and Ubuntu) are provided within the zPDT “package” due to slightly different library levels in these environments.

## 2.6.1 zPDT operational elements

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

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

zPDT installation also creates `/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 System z compatibility. The `man` directory contains normal Linux *man* pages for zPDT.

The first startup of zPDT creates a number of subdirectories (placed in the `z1090` subdirectory) in the user’s home directory.<sup>22</sup> Briefly, these subdirectories are as follows:

- ▶ `cards, lists`: May be used to provide input files to an emulated card reader or output from an emulated printer.<sup>23</sup> If not used, they are empty.
- ▶ `disks, tapes`: May be used to hold emulated disk or tape volumes, but these subdirectories are typically not used for anything. The emulated volumes are usually placed elsewhere, in other Linux file systems.
- ▶ `logs`: Used by zPDT to hold various dumps, logs, and traces. zPDT partly manages the contents of this subdirectory. The contents of this directory are important if it becomes necessary to investigate a zPDT failure.
- ▶ `configs, pipes, srdis`: Used for zPDT internal processing; do not erase or alter the contents of these small subdirectories.

Minor use of the `/tmp` file system occurs during zPDT installation, AD-CD installation, aws3270 device manager startup, and optionally for STP logs.

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

The emulator modules that provide System z functionality are not further described. They are not directly touched by the zPDT user. The device managers are described in 3.3, “Manager stanzas” on page 39. The syntax of the zPDT commands is described in Chapter 4, “zPDT commands” on page 51. Practical uses of zPDT commands, device managers, and `devmaps` are explained, at length, throughout the rest of this document.

## 2.6.2 Memory

The complete zPDT environment exists in Linux virtual memory. Linux is aggressive in allocating real memory frames to virtual memory pages and disk file data using its own

<sup>22</sup> When zPDT is started, a `z1090` subdirectory is created in the home directory of the user (if it does not already exist). The subdirectories discussed here are under the `z1090` subdirectory.

<sup>23</sup> Use of these directories is not required for emulated card reader or printer operation.

(Linux) judgment about what is the best use of real memory. The situation is complex when Linux caching of disk I/O is considered, and disk caching is a very important element of Linux performance.

We consider 8 GB of memory (in a PC) to be the *minimum* for zPDT usage, perhaps for z/VM, z/VSE, or Linux for System z. An 8 GB machine is usable for a very modest z/OS system. Memory might be much larger.

It is important to understand that zPDT exists in Linux virtual memory. We might *informally* say something like, “With an 8 GB machine we can allocate 1 GB to Linux and 7 GB to zPDT,” but such statements *must not be taken literally*. zPDT does not physically partition PC memory in any way. If we inspected the machine in this example at a random time, we might find 1.2 GB owned by the primary zPDT module, 0.2 GB owned by recognizable core Linux functions, 3.8 GB used for disk data cache, 0.2 used by various other processes (such as zPDT device managers and so forth) and the rest unassigned. A few seconds later, the usage statistics might be different.

We suggest that the PC memory size be *at least* 1 GB larger than the sum of all concurrent zPDT-defined System z memory. More is much better. A more typical arrangement might have PC memory at least twice the size of the defined System z memory. The primary goals are to (1) avoid Linux paging that stalls zPDT operation, and (2) to allow Linux to have an effective disk cache. *There is no easy way to directly manage either of these goals. They are indirectly managed by providing ample PC memory.*

### 2.6.3 Disk space

The disk space for the zPDT executable programs and control files is relatively small.<sup>24</sup> The disk space for emulated System z volumes is not small and some planning is needed. The space for emulated disk volumes can be calculated accurately, while 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 as follows:

3390 model	Approximate space required	Exact space required
3390-1	.95 GB	948,810,752 bytes
3390-2	1.9 GB	1,897,620,992 bytes
3390-3	2.8 GB	2,846,431,232 bytes
3390-9	8.5 GB	8,539,292,672 bytes
1 3390 cylinder		852,480 <sup>25</sup> bytes

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

Tape sizes reflect the size of the data written on the tape with a very small additional space (less than 1%) needed for awstape control blocks.<sup>26</sup> (Optionally, the awstape device manager can compress these files, often greatly reducing the amount of space used.)

### 2.6.4 LAN adapters

Both the base Linux and the z operating system can use a single Ethernet adapter (“NIC”) in the base PC. More than one adapter can be used, but this is usually unnecessary and can

<sup>24</sup> It is typically less than about 70 MB.  
<sup>25</sup> Each volume has an additional 512 bytes overhead.  
<sup>26</sup> The actual overhead is 6 bytes for each block written (including a tape mark, which counts as a block).

lead to routing complications. Wired or wireless adapters can be used, although care must be taken not to disrupt a wireless connection when used for 3270 sessions or CTC connections.

LAN usage is discussed in greater detail in Chapter 7., “LANs” on page 115.

LAN setup is often the most complex part of zPDT installation and use, especially when configuring both the base Linux LAN connections and the z operating system TCPIP connections. Some degree of LAN and TCP/IP understanding is required for any zPDT configuration beyond the most basic “local” 3270 operation.

## 2.6.5 Device maps

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

The following devmap illustrates a simple System z:

```
[system]
memory 5000m                # emulated System z to have 5000 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 zPDT programs that emulate various device types. The number after the device manager name is an arbitrary hexadecimal number (up to four digits) that must be different for each name statement.

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

## 2.6.6 Linux directory structure

A Linux user running zPDT has the following *default* directory<sup>27</sup> structure in Linux:

Directory path	Purpose
/home/<userid>/z1090/logs/	various traces are placed here
/home/<userid>/z1090/configs/	(internal 1090 functions)
/home/<userid>/z1090/disks/	emulated disk volumes
/home/<userid>/z1090/tapes/	emulated tape volumes
/home/<userid>/z1090/cards/	input to the emulated card reader
/home/<userid>/z1090/lists/	emulated printer output
/home/<userid>/z1090/pipes/	(internal 1090 functions)
/home/<userid>/z1090/srdis/	(internal 1090 functions)
/usr/z1090/bin	executable 1090 code, scripts
/usr/z1090/man	minor documentation
/usr/z1090/uim	identity manager files

Notice that different Linux userids would have different default 1090 directories and files. 1090 operation is sensitive to the Linux userid being used. The use of the default logs, lists, 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, based on the assigned device number, but 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 disk and tape subdirectories and these are typically empty.)

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

We suggest using a separate Linux file system for emulated disk volumes. This 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 book assume that all emulated I/O files are placed in the /z directory.<sup>28</sup> In our case, when we installed Linux, we created a separate partition (with a large amount of disk space) that is mounted at /z. We use this to hold all the emulated volumes. The cards, tapes, and lists directories, in the default directory path, are seldom used in typical operation.

<sup>27</sup> These names are subject to the discussion about the home directory. You should substitute the appropriate home directory name for the /home/<userid> portions of these names. A home directory could be almost anywhere in the root file system or in another file system. Our examples are based on the default form used by current Linux distributions.

<sup>28</sup> The mount point name, /z in our examples, is completely arbitrary.

## 2.6.7 zPDT control structure

The general structure of zPDT control files is shown in Figure 2-2.

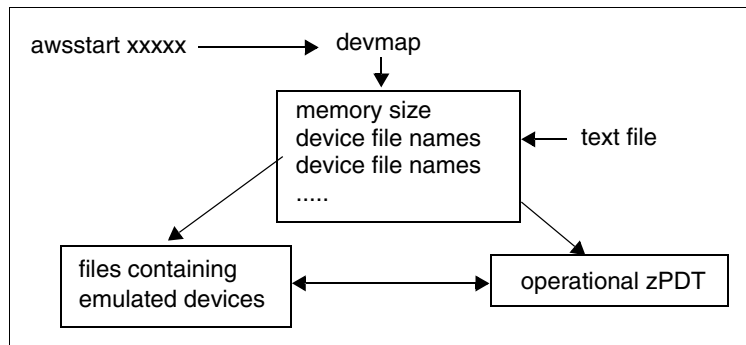


Figure 2-2 Control files: general structure

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

## 2.7 ISV zPDT and zD&T differences

The zPDT package available to independent software vendors (ISVs) and IBM internal users requires a Gen1 1090 token (or a remote license manager with an equivalent license). It will not function with a 1091 token. All zPDT functions are available with a 1090 token except that single tokens larger than an L03 (three licenses) are not available.<sup>29</sup>

All zPDT functions are available with a Gen1 1091 token (or a remote license manager with an equivalent license) but some functions require an additional license. The zD&T product requires a 1091 token and will not function with a 1090 token. Users of zD&T who require the coupling facility must purchase licenses features that enable that function.

The license terms and conditions for the two packages are quite different and are not addressed in this document. For complete licensing terms, contact your IBM representative, your zPDT supplier, or refer to the license documents supplied with your copy of zPDT.

This document primarily discusses Gen1 1090 tokens. However, other than the differences mentioned in this section, the document also applies to Gen1 1091 tokens (and equivalent remote license functions). The following table summarizes basic differences between ISV zPDT and zD&T.

Table 2-2 1090 and 1091 comparisons

ISV use - Gen 1 1090 tokens	zD&T use - Gen1 1091 tokens, or equivalent
1090 token only; 1091 token not usable (or equivalent remote licenses)	1091 token only; 1090 token not usable (or equivalent remote licenses, or a Gen2 software license)
Maximum of 8 CPs (with multiple tokens)	Maximum 8 CPs (with multiple tokens or a “large” token)

<sup>29</sup> However, more than one 1090 token may be used to provide more than three zPDT licenses.

ISV use - Gen 1 1090 tokens	zD&T use - Gen1 1091 tokens, or equivalent
Coupling Facility usage (under z/VM)	Coupling Facility usage (under z/VM) only with additional license feature
1, 2, or 3 licenses in token. No way to order larger tokens.	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 AD-CD system	Might have additions to the z/OS AD-CD system; 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 created in a subdirectory named z1090 in the Linux zPDT userid home directory. There is no /usr/z1091 or ~/z1091 usage.	
z/VSE and z/VM available (with proper license)	z/VSE not available. z/VM available for limited use only for use with the optional Coupling Facility license.
z/TPF not available at the time of writing.	z/TPF availability can be requested.
Installation and operational commands are as described in this book. (ITC offers additional interfaces with their uPDT package).	zD&T provides additional installation and operation programs and interfaces, not described in this book.

## 2.8 zPDT releases

There have been a number of 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 zPDT release. Other Linux distributions with libraries at an equivalent level (or later) might be used, although only the other listed distributions were tried by the developers.

### 2.8.1 Current release (GA10)

Table 2-3 Version 1 Release 10.

Characteristic	Version 1 Release 10
Date released	March, 2020
Initial zPDT driver level	55.xx
System z architecture level	z15
Linux levels used to build and test zPDT release. (The “official” levels.)	RHEL 7.7, SLES 12 SP3, Ubuntu 16.04.6 LTS
Informally tested Linux levels (Earlier levels should used with caution)	RHEL 7.x, 8.0 SLES 12 SP3, 15, openSUSE Leap42.3, 15.0, Ubuntu 16.04 LTS, 18.04.
Tested z/OS levels	2.2, 2.3, 2.4

Characteristic	Version 1 Release 10
z/VM used during development	6.4, 7.1 (not all functions)
Tested z/VSE levels	6.2
Tested Linux for System z level	<b>RHEL:</b> 7.1, 7.2, 7.4, 7.5, 7.6, 8.0 <b>SLES:</b> 12 SP1, 12 SP3, 11 SP3, 11 SP4, 15 <b>Fedora:</b> 23, 27, 28, 29 <b>Ubuntu:</b> 16.04.5, 18.04.1 <b>zKVM:</b> 1.1, 1.6, 1.1.2.9
Machines used for testing	Lenovo W540, P70; IBM System x 3850 M5, IBM System x 3650 M5
Virtual environments used during testing	VMware, KVM, Xen

Key elements include the following:

- ▶ The instruction set is at the z15 level. This represents the primary content of this zPDT release. An important variation involves use of the DEFLATE function (based on the DFLTCC instruction). The DEFLATE function is not included in the GA10.0 release. Please see the note in 12.2, “DEFLATE and zEDC emulation” on page 217 for more information about this function and about zEDC adapter emulation.
- ▶ This zPDT release has been successfully used with docker containers.
- ▶ CFCC level 24 is included.
- ▶ Crypto adapter level 7S is included. The format of the master keys<sup>30</sup> differs from earlier versions and zPDT does not automatically convert the older form to the newer form. You must initialize the new crypto adapter level with the same master keys previously used if you need to access previously encrypted data.
- ▶ MSA9 is not included in zPDT GA10.0.
- ▶ zPDT no longer supports general SCSI tape drives. Only Fibre-attached 3592 drives are supported. See Chapter 14., “Tape drives and tapes” on page 281 for more information.
- ▶ Various minor bugs have been resolved.
- ▶ Additional functions may become available in zPDT fixpacks.

## 2.9 Previous zPDT releases

Key characteristics of earlier releases are provided in the following table.

GA	Date	zPDT	Arch	Selected details
GA1	Oct. 2009	39.11	Z900 ALS3	Base: RHEL 5.2, openSUSE 10.3 Informal: Fedora, openSUSE 11 z/OS: 1.9, 1.10 Note: has 32-bit version
GA2	June 2011	41.21	z10 ALS3	Base: RHEL5.3, openSUSE 10.3, 11.1 Informal: openSUSE 11.1, Fedora z/OS: 1.10, 1.11 Note: pdsUtil, listVtoc, COD, MSA 3&4, 32-bit version

<sup>30</sup> On a “real” System z cryptographic adapter the master keys are securely held in the adapter. With zPDT, they are placed in the /~z1090/srdls directory of the Linux user who started zPDT. The format of this data is not documented.

GA	Date	zPDT	Arch	Selected details
GA3	March 2012	43.20	z196	Base: RHEL 5.4, SLES 11, openSUSE 11.2 Informal: openSUSE 11.3, 11.4, Fedora 12 z/OS: 1.11, 1.12, 1.13; z/VM 5.3, 5.4, 6.1; z/VSE 4.2, 4.3, 5.1 Note: remote license servers, devmap updates, CFCC17, LAN interface names, token RAS, 8 CP+speciality, 32-bit version no longer available, shared DASD, migration utility, RDzUT
GA4, 4.1	Dec 2012	45.18	z196, EC12	Base: RHEL 6.1, openSUSE 11.3, Informal: openSUSE 11.3,12.2,SLES 11 SP2, Fedora 15,17 z/OS: 1.12, 1.13, z/VM 6.1 some 6.2 Note: many new instructions, zBX, virtualization, integrated consoles, z1090term, CEX4C, CFCC18
GA5	Feb 2014	47.xx	EC12 GA	Base: RHEL 6.x openSUSE 11.3, SLES 11 SP2 Informal: Fedora 17,19, openSUSE 11.4, 12.1 z/OS: 1.12, 1.13, 2.1 z/VM: 6.2, some 6.3 Note: CEX4, CPACF updates, CFCC19, SCSI 359x
GA6	March 2015	49.xx	z13	Base: RHEL: 7.0, SLES 11 SP3 Informal: openSUSE 13.1, SLES 11 SP3, Fedora 20 z/OS: 1.13, 2.1; z/VM 6.2, 6.3; z/VSE 5.1, 5.2 Note: many new instructions, CEX5S, CPACF updates, STP, CFCC20 SL16, r/o DASD, KVM, zBX
GA7	March 2017	49.xx	z13 GA2	Base: RHEL6.3 - 7.1, SLES 11 SP2, Ubuntu 16.04 Informal: Fedora 25, Leap42.1, SLES 11 SP3 z/OS: 1.13, 2.1, 2.2; z/VM 6.2 - 6.4; z/VSE 6.1 Note: Ubuntu base, z/TPF, software license server, CFCC21, up to 2048 devices, zBX dropped
GA8	Dec 2017	51.xx	z14	Base: RHEL 7.3, SLES 12 SP1, Ubuntu 16.04, Leap42.1 Informal: Leap42.1, openSUSE13.1, Fedora25 z/OS: 2.1, 2.2, 2.3; z/VM 6.2 - 6.4; z/VSE 6.1 Note: New instructions z14, "free" zIIPs, CFCC22, revised awssckd (2017, MIDAW), jumbo frames, I/O counts via awssstat
GA9	March 2019	53.xx	z14 GA2	Base: RHEL 7.5, SLES SP3, Ubuntu 18.04 Informal: Leap42.3, Leap15, Fedora 27,28,CentOS7.0, 7.4 z/OS: 2.2, 2.3, 2.4 Note: special tape CCWs, LPAR name, console file, CFCC23, zEDC, channel measurements, layer 2 fixed, 64-bit support for gen2 licenses

More extensive lists of the zPDT changes in the various releases is given below. Each release typically includes performance improvements; these are not indicated below.

#### zPDT Release 2:

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

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

#### zPDT Release 3:

- The relevant instruction set for the System z 196 processor is included. This is a major change for the base zPDT element.
- A remote license server allows the USB keys to be installed in a central location. Multiple standard USB keys may be used (each with a maximum of three CP licenses) or nonstandard keys containing more licenses. Associated with this function is a Unique Identity Manager (UIM) that provides the same consistent serial number for the System z CPs in a given Linux machine.
- The device map (`devmap`) used to define an instance of zPDT operation may now include Linux commands, with a method to control the timing of these commands. This function may be used to automate zPDT startup among other uses. In addition, environmental variables, `include` statements, and `message` statements are permitted in the `devmap`.
- The cryptographic adapter functions provided by zPDT are now at the Crypto Express 3 level.
- z/OS releases earlier than 1.12 might require the fixes for APAR OA29839 to be applied.
- The license server function associated with token processing has been expanded to add significant security options.
- A new level of the Coupling Facility code is included, which is level CFCC D93G R17 SL4.8. This CFCC level is considerably larger than the CFCC included in the previous zPDT release and a larger z/VM guest is needed to use it.
- The Linux `/etc/profile.local` and `/etc/profile` files no longer require modification.
- The handling of LAN interfaces has been expanded to handle the new LAN interface names being used in later Linux releases.
- The zPDT `stop` and `start` commands are extended to include `stop all` and `start all`.
- New RAS functions improve access to the USB key in rare cases where problems have been reported. The methods for starting the token interfaces during Linux booting have been enhanced.
- The maximum number of CPs (or the total of CPs, zIIPs, zAAPs, and IFLs) for a zPDT instance is now specified as eight.
- The use of USB 3 ports (for the USB key) is now supported.
- The 32-bit version of zPDT, previously available only within IBM, is no longer available.
- The `SecureUpdateUtility` command must be run from the `/usr/z1090/bin` directory and must be run as `root`.
- Emulated DASD (3380, 3390) may be shared between instances of zPDT on the same PC. The performance of the zPDT locking involved in this sharing has been enhanced.

- zPDT includes a migration utility that may be used to copy 3390 volumes from a remote z/OS or z/VM system. The z/OS version of this utility previously included an *automatic restart* function that attempted to restart at the point of failure if a migration transfer was interrupted. This function has been removed. If a volume migration is disrupted, it must be started again.
- When zIIPs or zAAPs are defined in a device map, at least one “cp” definition must precede the zIIP and/or zAAP in the processors statement.
- The output of the **token** command has been expanded to provide both zPDT license information and CP serial number information.
- Linux environmental variables may be used in device map specifications.
- Several minor commands have been added to permit an installation to administer zPDT tokens and license server configurations without switching to the Linux root userid.
- RDzUT customers may use more than three zPDT CPs, assuming sufficient zPDT licenses are available.
- The specification of **ulimit -c unlimited** for the zPDT operational environment might be reconsidered. This might be relevant for very large zPDT instances with, for example, 32 GB and larger System z storage specified.

#### zPDT Release 4 (and a fixpack):

- The relevant instruction set for the System z EC 12 processor is included. This is a major change for the base zPDT element. This includes significant new EC 12 functions:
- 2 GB Page Frames
- The flash memory function of EC 12 systems is *not* provided by zPDT at this time.
- 1090 and 1091 tokens may no longer be used interchangeably. The IBM Rational license manager may be used in conjunction with a 1091 token.
- Tokens with more than 3 zPDT licenses may be used with 1091 systems that are enabled for such usage.
- Two general virtualized environments may be used with zPDT. These are discussed in Chapter 18, “Virtualization and containers” on page 325.
- Additional command scripts (**aws\_bashrc** and **aws\_sysctl**) are available to simplify zPDT installation. Also, there is now a **1091ver** command to match the older **1090ver**.
- The integrated consoles (3270 and ASCII) that are available with an HMC may be emulated with zPDT.
- A new command, **z1090term**, provides an ASCII console that can be connected to the integrated ASCII console interface.
- 3390 (and 3990) emulation has been upgraded to the level required for z/OS 2.1 (expected to be released in 2H2013).
- The remote license server that allows the USB keys to be installed in a central location has been improved.
- The cryptographic adapter functions provided by zPDT are now at the Crypto Express 4 (EC 12) level (CEX4C).
- A new level of the Coupling Facility code is included, which is level CFCC Level 18.
- zPDT includes a migration utility that may be used to copy 3390 volumes from a remote z/OS or z/VM system. This has been updated to function with older DASD on the “real” System z.

#### zPDT Release 5

- The major new element in this release is the processing of encrypted and protected AD-CD releases. This function restricts the usage of IBM System z operating systems, packaged in the AD-CD format, to zPDT systems.
- New cryptographic adapter levels corresponding to CEX4:
- Export triple DES (TDES) key under an AES transport key
- Diversified key generation cipher block chaining (CBC) support

- Initial PIN encrypting key (IPEK) support
- Remote key export (TKX) key wrapping method support
- Integration of User Defined Extensions (UDX) into CCA
- (CEX4 also matches CEX3 and might be identified as CEX3 by ICSF)
- CP Assist Crypto Facility (CPACF) updates
- New installation commands and techniques for AD-CD z/OS 2.1 involving encrypted distribution of z/OS volumes
- New token license update commands
- New support for SCSI-attached 359x tape drives
- CFCC 19 Coupling Facility level

#### zPDT Release 6

- New instructions corresponding to the architecture of the IBM z13 series.
- CPACF supports MSA5 (random number generation)
- Vector Facility (SIMD) with 139 new instructions and 32 registers of 128 bits.
- New cryptographic adapter levels corresponding to CEX5S:
- Maximum number of domains (per emulated adapter) is 16.
- Format Preserving Encryption instructions
- IBM DCP® Assist Crypto Facility (CPACF) updates.
- Support for the System Time Protocol (STP). This involves Linux daemons running on multiple zPDT hosts (multiple PCs) and is typically used in a basic sysplex environment.
- CFCC 20 Coupling Facility, service level 16
- Support for read-only DASD volumes.
- Support for shared DASD volumes (across multiple PCs) in an NFS environment.
- zPDT has been tested in a virtual environment provided by KVM.
- zAAP speciality processors may be configured with this zPDT release although, strictly speaking, they should not exist in a z13 system. The user must determine the usefulness of zAAPs in this situation.

#### zPDT Release 7

- Cryptographic adapter level corresponds to the relevant features in current System z.
- New zPDT commands to create and inspect labels of emulated tape volumes.
- Support for operation under Ubuntu Linux, but only when a remote zPDT license manager is used.<sup>31</sup> At the time of writing, a Ubuntu base Linux had not been tested with more than three zPDT CPs.
- Fixes for SafeNet operation with recent Linux levels.
- z/TPF operation (limited availability)
- E-mail notification of pending license expirations.
- A “software” license server (with no hardware token involved) for limited use.<sup>32</sup>
- Support for a new type of zPDT license servers and tokens.
- Command-line interfaces to set remote license server parameters (as opposed to interactive or graphic interfaces).
- Improved internal design to reduce “hacking” exposures.
- CFCC 21 Coupling Facility.
- The maximum number of zPDT devices has been expanded from 1022 to 2048. See 13.22, “Many zPDT devices” on page 267 for a discussion of this topic.
- zAAP speciality processors may be configured with this zPDT release although, strictly speaking, they should not exist in a z13 system. The user must determine the usefulness of zAAPs in this situation.
- zPDT is no longer tested for zBX usage.

<sup>31</sup> This restriction existed at the time of writing, due to the unavailability of a Ubuntu version of the SafeNet token support modules.

<sup>32</sup> At the time of writing, this function is only for zD&T use.

## zPDT Release 8

- The architecture level (that is, the System z instruction set) corresponds to IBM z14 machines. This involves significant enhancements to zPDT. This includes defaulting to z Architecture for IPL operation.
- A non-standard option allows IPLing in ESA/390 mode, but operation in z14 mode otherwise. See 13.4, “cpuopt statement” on page 255 for details.
- zPDT licenses are no longer required for zIIPs, although the zIIPs count toward the maximum of eight processors in a zPDT instance. zAAPs and IFLs require zPDT licenses. (In principle, zAAPs no longer exist with z14 machines.) The number of zIIPs cannot exceed the number of CPs.
- Coupling functions are updated to the CFCC 22 level.
- The awsckd device manager (that emulates 3380 and 3390 operation) has been reworked.
- The awsckd device manager emulates an IBM 2107 control unit in place of the 3990 control unit in prior releases and newer CCW operations are handled correctly.
- MIDAW operation is included
- I/O counts are available with the **awsstat** command. This data can be especially useful when emulated disk volumes are spread among devices with very different performance characteristics.
- The awsosa device manager corresponds approximately to the OSA-Express5 level.
- The awsosa device manager supports promiscuous mode Ethernet, as needed for KVM operation.
- Jumbo frames (up to 8992 bytes) are supported.
- Deprecated commands/scripts are removed: **ldk\_server\_config** (replaced by **clientconfig**), **LDKc\_setup.sh** (replaced by **gen2\_init**).

## zPDT Release 9 (including earlier fixpacks):

- Two special CCWs are available for awstape management. These CCWs allow suitable programs to query or change the Linux file name (with the file in awstape format) associated with an emulated tape drive. Details are available in “Use special CCWs to manipulate tape volume” on page 275.
- A devmap option allows the zPDT instance name (the “LPAR name”) to be specified. Using this option allows the Linux userid under which zPDT runs to have more than eight characters.
- A devmap option may be used to specify the a Linux file to receive messages that the z operating system routes to the HMC “hardware” console. The default routing is to the Linux command window that was used to start zPDT.
- The **adstop** and **storestop** commands have been removed. (The effectiveness of these commands was questionable due to JIT optimization.)
- The sizes of two Linux kernel parameters set by the **aws\_sysctl** script have been changed. The shmmax size is now 36,000,000,000 and the shmall size is now 24,000,000. These are twice the previous values and are in recognition of increasingly large PC memory sizes.
- CFCC level 23 is included.
- A restricted for of zEDC emulation is available for z/OS (but not for z/VM). See “DEFLATE and zEDC emulation” on page 217 for details.
- Selected channel/device measurement data is available through the architected interface.
- The translate table for **awsprt** (for EBCDIC to ASCII conversion) is improved.
- The **gen2\_init** command (used for Gen2 licenses) includes an option to remove the Gen2 license modules.
- A long-standing problem with layer 2 VSWITCH has been corrected.
- The Gen2 license server and/or client no longer require 32-bit Linux support libraries. (The Gen1 license server (for 1090 and 1091 tokens) still requires 32-bit support.

- The **a1cckd** utility now supports definition of emulated 3390 volumes with exactly 65520 cylinders. (This capability was missing in zPDT GA8.)





# Devmaps

In this chapter, we provide reference information for zPDT device map entries for the System z processors and I/O device managers. Information and guidance for using these device managers is found throughout this book.

## 3.1 Device maps

A device map (devmap) consists of a system stanza, optional adjunct processor, zEDC (if appropriate), System Timer 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 document.

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

Create a devmap file in lowercase,<sup>1</sup> 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 shown in the descriptions below are part of the syntax and must be entered as shown.

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

---

<sup>1</sup> This is not required by some elements of a devmap, which ignore upper or lowercase differences. Not all devmap elements do this. To avoid problems, we suggest using lowercase for everything (except for Linux file names, which are case-sensitive).

## 3.2 System stanza

A [system] stanza might look like this:

```
[system]
memory 16G          # define 16 GB memory for System z
processors 1         # number of CPs
3270port 3270       # specify unique IP port number for aws3274
expand 0m           # no expanded storage
ipl 0A80 "0A8200"    # automatic ipl control (optional; not recommended)
cpuopt alr=on       # optional function (defaults to on)
command 2 x3270 localhost:3270 #Linux command executed via the devmap
command 2 x3270 -model 3279-5 localhost:3270 #various x3270 options available
```

The memory statement specifies the size of the System z memory to be emulated for zPDT operation. The number specified must be smaller than the maximum shared memory value specified for Linux; this is set by the kernel `.shmmx` parameter in Linux.<sup>2</sup> For z/OS the memory value should normally be at least 4G; we typically use 8G-12G but it might be much larger than this.<sup>3</sup>

You must have a processors statement unless your devmap is for a group controller. (See Chapter 10, “Multiple instances and guests” on page 197 for more information about group controllers.) The processors statement specifies the number of System z processors to be used in this instance. The default is one. The processors statement is also used to indicate the use of speciality PUs as in the following examples (where `cp` indicates a normal, general-purpose CP):

```
processors 3          # three CPs. Assumed “cp” type
processors 3 cp cp ziip # two CPs and one zIIP; 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)<sup>4</sup>, which (excluding ziip processors) cannot exceed the number allowed by the number of licenses allowed by the zPDT token. The processors default to CPs; if you want speciality processors, list them after the number as shown in the examples. The processor types are `cp`, `ziip`, `zaap`, and `ifl`. If zIIPs or zAAPs are specified in the processors statement, at least one CP must be listed first.<sup>5</sup> Speciality SAP processors are not used and Coupling Facilities (CFs) are not specified in the devmap.<sup>6</sup>

The zAAP processors might not be useful in the z13 (or later) environment created by zPDT GA6 or later. Starting with zPDT GA8, zIIP processors do not require a zPDT license although they are included in the maximum of eight processors for a zPDT instance and are included when determining the minimum number of PC cores needed.

The expand statement specifies the size of expanded storage for the System z machine. This is optional. z/OS no longer uses expanded storage. However, some older releases of z/VM still use it.

<sup>2</sup> This kernel variable is specified by the instructions Chapter 5, “zPDT installation” on page 95.

<sup>3</sup> The memory size may also be specified in megabytes by using a “M” suffix.

<sup>4</sup> The number of processors for an instance has a maximum value of 8. This is usable only if multiple tokens are used or nonstandard “high capacity” tokens are used, and may be limited by zPDT license terms and conditions.

<sup>5</sup> The first processor type listed becomes the IPL processor; zIIPs and zAAPs cannot handle an IPL.

<sup>6</sup> zPDT uses z/VM to provide emulated CFs.

The `3270port` statement specifies a port number to be used by the base Linux TCP/IP for the `aws3274` device manager. This must be an unused port on the base Linux and is typically a number greater than 1024. 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 System z.

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

The `cpuopt` statement specifies optional parameters for the CPs. The following statement should be used only by zD&T customers who have the optional Coupling Facility feature with their license.

```
cpuopt zVM_CouplingFacility      (no blanks in operand)
cpuopt zVM_Coupling              (this abbreviation can be used)
```

In effect, the `zVM_CouplingFacility` function is always present for ISV zPDT systems. Other `cpuopt` parameters produce non-standard configurations and are described in 13.4, “`cpuopt` statement” on page 255.

**Important:** 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`, zPDT startup will fail to acquire licenses.

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

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

The phase number is a digit from 1 to 4:

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

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

Additional system stanza options include these:

```
[system]
...
rdtserver 27000@our.server.acme.com    # Rational License Server and port
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 `rdtserver` statement is used only with a zD&T system. It points to an IBM Rational License Server used to supplement the zPDT license.<sup>7</sup> A Rational License Server is not the

same as a zPDT remote license server and is not required for basic zPDT 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 rdtserver can be specified by using a colon as the separator:

```
rdtserver 27000@server1.com:7777@server2.com
```

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

The 3270 terminal session associated `int3270port` function *probably* should<sup>9</sup> 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 AD-CD z/VM distribution.

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

By default, zPDT uses the Linux userid that was used to start zPDT as the instance name (the “LPAR” name). An LPAR name cannot be longer than 8 characters and this restricts the Linux userid to not more than 8 characters. However, the `system_name` option (maximum 8 characters) can be used to specify the LPAR name, removing the length restriction on the Linux userid used to start zPDT. The specified name is automatically converted to upper-case EBCDIC.

By default, zPDT sends output directed to the “HMC hardware console” to the Linux command window that was used to start zPDT. (If this command window is closed, the output is not displayed.) By default, this output (along with any **oprmsg** commands) is written to a zPDT log file. The `oprmsg_file` option specifies a Linux file that is to receive such messages;<sup>10</sup> If an `oprmsg_file` is specified the messages are no longer written to the default zPDT log file. The file name is case sensitive. This option has several implications:

- ▶ The messages are available in the specified Linux file whether or not the original Linux command window is open.
- ▶ If `oprmsg_file` option is used, the default zPDT log file will not contain the messages written to the specified `oprmsg_file`.
- ▶ Typically, this option is used in conjunction with automation (running under Linux) that generates **oprmsg** commands sent to the System z. The responses, along with other operating system output, are placed in the specified Linux file where they can be inspected by the automation programs.<sup>11</sup>

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

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

<sup>7</sup> Contact your IBM marketing representative for more information about Rational licenses.

<sup>8</sup> See the command descriptions in Chapter 4, “zPDT commands” on page 51.

<sup>9</sup> 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.

<sup>10</sup> This function is described in more detail in 13.7, “zPDT log files” on page 258.

<sup>11</sup> zPDT does not provide any such automation functions or examples.

```
[system]
memory 8G
processors 2
3270port 3270
command 2 x3270 -model 4 localhost:3270
command 2 x3270 -model 4 localhost:3270
command 4 echo '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 zPDT is shut down.

A devmap has several additional features <sup>12</sup> as follows:

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

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

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

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

include dasd.def

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

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

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

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

```
$ export SIZE=6500m
```

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

<sup>12</sup> These features were added for special purposes. We have not seen much usage by typical zPDT users.

The **include** function in the example operates as you might expect. The file specified is logically inserted into the devmap at the point shown. The operand of the **include** function can specify a full Linux path name; if a simple name is specified, it is assumed to be in the current directory. The file name specified cannot contain blanks. The name could be an environmental variable instead of a file name, for example **include \$(fileVAR)**. If the specified environmental variable is not defined, the **include** function is skipped.

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

It is very unlikely that all the [system] stanza options would be used at one time, but here is 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 8 characters
oprmsg_file /tmp/bill/messages
#ipl 0A80 "0A8200"                           (not recommended. Commented out here)
cpuopt alr=on,zVM_Coupling
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.

### 3.2.1 Adjunct-processor stanza

The zPDT system optionally provides emulation of the System z cryptographic adapter.<sup>14</sup> The release level of the cryptographic adapter varies with the zPDT release level. The basic devmap format for this emulation is as follows:

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

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

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

This indicates that the instance is using domain 2 in cryptographic coprocessors 0 and 1. See Chapter 17, “Cryptographic usage” on page 307 for more details.

<sup>13</sup> Other required changes to the `.bashrc` file are described in Chapter 5, “zPDT installation” on page 95.

<sup>14</sup> Do not confuse this with the cryptographic instructions, which do not require any special devmap statements.

### 3.2.2 System timer protocol stanza

The system timer protocol (STP) function is described in Chapter 20, “Server Time Protocol (STP)” on page 337. The Linux daemons associated with STP must be started before using a devmap containing a 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 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 may also include a `LEAPSECONDS` statement, not shown in this example. This is further described in Chapter 20, “Server Time Protocol (STP)” on page 337.

### 3.2.3 zEDC stanza

zPDT GA9 provides limited zEDC emulation, which is described in 12.2, “DEFLATE and zEDC emulation” on page 217. The required stanza can look like the following:

```
[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 zPDT use.

## 3.3 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 this could be any of the supported device managers. The device manager name is followed by an arbitrary hex number (up to four digits, different for each name statement)<sup>15</sup>. The name statement is followed by as many device statements as needed. 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.
  - Additional optional parameters, such as these:
    - `--path=xx` to specify an emulated CHPID number.

<sup>15</sup> 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.

- `--pathtype=xxx` to specify an emulated CHPID type (usually EIO).
  - `--compress` to specify compressed awstape generation.
  - Various optional *tunnel* parameters for OSA operation.
- For device statements:
- The constant “device” starting in the first column.
  - The device number (“address”) to be used, expressed in hexadecimal. This may be three or four digits.
  - The device type, such as 3390. This must specify a correct device type for the device manager.
  - The control unit type associated with the device, up to four characters. (This parameter is not used for anything at this time, 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.
    - Additional parameters for OSA operation.

At this time, 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<sup>16</sup> for most device managers. In very rare cases it may be desirable to change these values. This can be done with the `--path` and `--pathtype` operands on a name statement, 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 may be used. A maximum of 256 devices may be listed in a stanza, where multiple devices are not limited by characteristics of the emulated control unit. The device numbers (addresses) assigned to each device need not be sequential or in any particular order.

### 3.3.1 The awsckd device manager

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

```
[manager]
name awsckd 4321 [--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 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 **a1cckd**

<sup>16</sup> EIO is a special CHPID type for Emulated I/O. zPDT users normally do not need to specify this anywhere.

command, or copied from media that originated on a system where the file was initially created with **a1cckd**. Each emulated volume is a single, separate Linux file.

The third operand of a device statement is a control unit type. This information is not currently used by zPDT, but it is a positional operand that must be specified. Starting with zPDT GA8, **awsckd** emulates IBM 2107 control units. You can specify 3990, 2107, or anything else up to four characters.

The most common CKD devices are 3390 units. Standard 3390s (models -1, -2, -3, and -9) may be used, or a variable number of cylinders may be used. The maximum size for a normal 3390 is 64K-1 cylinders; however, zPDT supports extended address volume (EAV) 3390s.

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

Device statements may omit a file name. In this case the indicated unit (3390 at address 0AA8 in the example) exists, but has no volume mounted. A volume (which is a Linux file in the appropriate CKD format) may be mounted (or dismounted) while zPDT is operational, providing dynamic changes to the DASD environment without stopping 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 do these 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 System z channel program.

Much more is involved in sharing System z volumes and the **--shared** option is only one element involved. Use this option when multiple zPDT instances (in the same Linux) are sharing DASD volumes and when separate zPDT systems (on separate Linux bases) are sharing DASD through a Linux shared file system. Proper serialization, as seen by z/OS, is essential for shared DASD and implementing such serialization typically involves z/OS GRS.<sup>17</sup> The **--shared** option is *not* used when sharing DASD volumes among multiple z/VM guests.

### 3.3.2 The **awsfba** device manager

The **awsfba** device manager provides emulation for FBA disk devices<sup>18</sup> (as used by z/VM and z/VSE).

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

**awsfba** devices (volumes) must be created before they can be used. This is done with the **a1cfba** utility. This device manager *does not* support the more recent Fibre open system FBA devices for z/OS. It is unfortunate that there are two unrelated uses of FBA terminology.

zPDT development performs only minimal testing for these FBA devices. We recommend using CKD disks unless there is a specific need for FBA disks.

<sup>17</sup> Global resource serialization (GRS) is a basic element of a System z sysplex configuration.

<sup>18</sup> These had IBM type numbers such as 3370, 9332, 9335, and 9336.

### 3.3.3 The aws3274 device manager

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

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

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

```
[manager]
name aws3274 C700                # C700 is an arbitrary CUNUMBR
device 0700 3279 3274
device 0701 3279 3274 L701
device 0702 3279 3274 L702
device 0703 3279 3274 TSO
device 0704 3279 3274 TSO
device 0705 3279 3274 TSO
device 0706 3279 3274
device 0707 3279 3274
device 0708 3279 3274 IMS
device 0709 3279 3274 IMS
device 070A 3279 3274 IMS
device 070B 3279 3274 IMS
device 070C 3279 3274
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 TN3270e client connections to the device. This operand is known as an LUsername, 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 actually passing it to VTAM.) The LU names may have a maximum of 11 characters.

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

- ▶ The LUsername is not case-sensitive.
- ▶ If an LUsername is specified by the TN3270e client, then a free device with the matching LUsername is used.
- ▶ If no LUsername is specified by the TN3270E client, the next free device in the list is used, whether or not it has an associated LUsername.
- ▶ If there is no free device to match the specified LUsername, 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 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 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 our x3270 client is on a remote machine connected to a private LAN that includes the zPDT system. In case 1, the user is connected to the next available 3270 session (in the devmap list). In case 2, the client is connected to the next free device with LName TSO. In case 3, the client is connected to the single device with LName L702, provided that device is free at this time. The fourth example illustrates that the same LName 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 LNames. 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 LName reflect the device address (device number).

The devmap for an AD-CD z/OS system might be defined like this (and this is the most common example for zPDT users):<sup>19</sup>

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

Connections take the next free terminal in the devmap list if no LU conditions are specified. This can be useful if the first terminal in the devmap is the MVS console<sup>20</sup> and the next terminal is a suitable TSO address. In this case, without specifying any LU names, the first x3270 session will be the MVS console and the second will be a TSO session (or CICS or some other VTAM application). As a practical matter, we have observed that very few 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 available for Linux have been tested for this usage.<sup>21</sup> The TN3270e client might operate on the machine running the 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 System z operating system.

<sup>19</sup> You might notice that the third operand in the aws3274 examples is sometimes 3174 and sometimes 3274 in the examples. This operand in device statements is not processed and can be any four character value. We suggest a meaningful value only for documentation.

<sup>20</sup> The AD-CD z/OS systems have always defined the MVS console at address 700.

<sup>21</sup> The aws3274 device manager sends an attention signal to the host when a session is first connected. In some cases, such as when connected to the VTAM unformatted system services function, this may prompt a full buffer read by the host software. If the TN3270e session buffer is not formatted for this buffer read, the host may display an "Unsupported Function" message. Simply clearing the TN3270 screen should resolve the situation. Some TN3270e emulators encounter this situation and others do not.

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

### 3.3.4 The awstape device manager

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
```

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

The `maxlength` parameter is optional. If a `maxlength` value is specified, the device manager signals end-of-tape after the specified number of bytes has been written. (z/OS would then probably write trailer labels and call for another tape mount.) If `maxlength` is not specified, then the maximum tape 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 four million tape blocks for 3480 and 3490 device types. The device signals end-of-tape just before this limit is reached. This limit exists for both reading and writing tapes.
- ▶ Device types 3420 and 3590 do not have specific limits.

The `--MNTCCWS` parameter enables use of 4B and 54 CCWs used to manipulate the Linux file name associated with the emulated tape drive. This is described in “Use special CCWs to manipulate tape volume” on page 275.

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

The proper responses for hardware compaction (IDRC) are emulated, although tape data is not actually compacted by this method. The `awstape` data may be optionally compacted by the `awstape` device manager. This is controlled through a `devmap` or an `awsmount` parameter. The compaction format is unique to zPDT `awstape`. The default uncompact 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 labelled tape. (An “empty” labelled tape can be created with the zPDT `aws_tapeInit` command.)

### 3.3.5 The awsosa device manager

The awsosa device manager emulates various OSA-Express functions, as used by System z TCP/IP or SNA.<sup>22</sup> Currently the emulation level is OSA-Express5s. Two manager formats are used:

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

[manager]
name awsosa 2345 --path=A0 --pathtype=OSD [--interface] [--tunnel_intf=y]
  [--tunnel_ip=10.1.1.1] [--tunnel_mask=255.0.0.0]
device 404 osa osa
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.<sup>23</sup> 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. That is, both OSA and Linux can share the adapter for connection to external TCP/IP systems, but they cannot communicate with each other.<sup>24</sup> A tunnel interface (which is similar to another Ethernet adapter) is necessary for direct communication between the underlying Linux system and the System z OSA operation.

**Tip:** See “LANs” on page 115 for details about using the emulated OSA functions.

The `--path` operand specifies a CHPID number. The correct number is determined with the `find_io` command. For these examples we assume the CHPID for wired Ethernet is F0 and the CHPID for a tunnel interface is A0. The `--pathtype` is OSD (for QDIO) or OSE<sup>25</sup> (for LCS or non-QDIO). In some cases the `find_io` command does not provide a CHPID (path name) for a LAN interface; 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, “LANs” on page 115. An example of using the `--interface` parameter might be:

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

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

The z/OS device type should be OSA, as seen in the z/OS IODF (and when displaying devices on the MVS console).<sup>26</sup>

When used in OSE mode, the OSA interfaces are associated with OAT<sup>27</sup> definitions that specify how each interface is to be used. Please note that OSA/SF (the z/OS program to manage OATs) is not available with current z/OS releases; the partial replacement is the

<sup>22</sup> zPDT SNA usage has not been tested by IBM and no support is available for it.

<sup>23</sup> If no IP address is specified for a tunnel interface it will default to 10.1.1.1.

<sup>24</sup> This odd limitation is a characteristic of current Linux implementations.

<sup>25</sup> We do not recommend OSE use unless you have a special need for it.

<sup>26</sup> Older z/OS systems may use CTC device definitions for these interfaces, especially when they are used for TCP/IP. These definitions should be replaced with device type OSA.

<sup>27</sup> An OAT is an OSA Address Table.

**QUERYINFO** command, which is *not* supported with zPDT. SNA usage would require CHPID type OSE, although there is no zPDT support for SNA usage.

The limits in Table 3-1 apply to OSA-Express emulation.

Table 3-1 OSA-Express limits, per port

Maximum OSAs (and maximum OSA CHPIDs)	4
Maximum home addresses (IPv4 + IPv6 + DVIPA) per OSA port	64
Maximum IPV6 addresses	32
Maximum multicast addresses (IPv4 + IPv6)	64
ARP table size	256
IP stacks per port (OSD or OSE)	16
SNA PUs per OSA-Express port ( <i>SNA is not supported for zPDT</i> )	512
OSE subchannels per stack	2
OSE or OSD maximum devices	48
OSE IP stacks per OSA port/CHPID	16
OSD subchannels per stack	3
OSD subchannels per OSA/CHPID	48
Is OSA/SF or QUERYINFO available?	No
Is Generic VLAN Registration Protocol supported? (GVRP)	No

### 3.3.6 The awsrdr device manager

The awsrdr device manager emulates a 2540 card reader. Only one awsrdr device may be configured for an instance of zPDT operation. Typically, the emulated card reader is used to submit jobs to the operating system.<sup>28</sup> If we assume this to be z/OS, then JES2 or JES3 should be configured with a “hot” reader.<sup>29</sup> The traditional address for a 2540 is 00C, and we use this in our examples.

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

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

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

<sup>28</sup> In principle, we could directly allocate the card reader to a job using the appropriate DD statement. We did not try this.

<sup>29</sup> The term “hot reader” means there is always a read outstanding for the card reader. As soon as an operator places cards in the reader, JES begins reading them.

The `/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 may contain binary information. A card reader uses fixed-length records (80 bytes) but a Linux text file has variable length records terminated with an NL character.

The conversion rules are as follows:

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

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

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

### 3.3.7 The `awsprt` device manager

The `awsprt` device manager emulates a 1403 or 3211 printer. FCB functions<sup>30</sup> are supported for 3211 emulation, but UCS functions for a 1403 are not supported. A fixed translation table is used to convert EBCDIC to ASCII. The device manager automatically inserts NL characters between output records. Unprintable characters are translated to blanks and no *unit check* is generated for these.

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

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

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

The `awsmount` command may be used to close the existing output file and open a new output file. The previous output file is closed properly, and is then available for display or printing under Linux. “Local printing” on page 233 provides more information about printing output with the `awsprt` device manager.

---

<sup>30</sup> FCB refers to the Forms Control Buffer.

### 3.3.8 The awscmd device manager

This device manager provides a “device” that appears to System z software as a tape drive. Its function is to send a command (and data) to the underlying Linux and then receive the output from the Linux command. Any Linux command may be sent, including those that could destroy the Linux system.<sup>31</sup> Obviously, this device manager should be used with care and might not be appropriate for a zPDT environment that can be accessed by untrusted users.

Configuration is similar to other device managers:

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

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

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

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

#### Restrictions

The command 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 will appear to be busy while Linux is executing the command. Any Linux command that creates substantial delays (of many seconds) might cause I/O timeout errors to be generated in z/OS.<sup>32</sup>

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 (^)

<sup>31</sup> The Linux commands are executed with the authority of the userid that started zPDT operation.

<sup>32</sup> Prior version of awscmd had a timeout function that limited the time allowed for the Linux command. This timeout check has been removed at customer requests.

- ▶ Colon (:)
- ▶ Double quotation marks (")
- ▶ Less than (<)
- ▶ Greater than (>)
- ▶ Question mark (?)

An extended example of `awscmd` usage is in Chapter 11, “The `awscmd` command” on page 207.

### 3.3.9 The `awsscsi` device manager

The `awsscsi` device manager emulates a mainframe tape drive using a SCSI tape drive. The specific adapters and tape drives supported are discussed in Chapter 14, “Tape drives and tapes” on page 281. 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. This must be given as a `/dev/sgx` name, and not as a `/dev/stx` name. The differences are complex; Chapter 14, “Tape drives and tapes” on page 281 describes methods for determining the correct `/dev/sgx` name. The SCSI tape drive appears as a 3420, 3480, 3490 or 3592 device to the System z software.<sup>33</sup> The `awsmount` command may be used with SCSI tape devices.

### 3.3.10 The `aws3215` device manager

The `aws3215` device manager provides emulation of a 3215 console, using `devmap` parameters such as these:

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

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

### 3.3.11 The `awsoma` device manager

The `awsoma` device manager is used to read CDs or DVDs<sup>34</sup> written in a special format known as OMA. This is for input only; it is not possible to write to an `awsoma` device. In earlier days, z/VM and z/VSE were available in OMA format; some Linux distributions for System z may use this format.

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

The variable portion of the device statement (after the second `oma`) must be in a specific format, with two names separated by a comma or semicolon. There must be no blanks

<sup>33</sup> Remember that IBM 3490 tape units have their own characteristics. One of these is a maximum block count of approximately 4 million (a 22-bit number).

<sup>34</sup> It is possible to have OMA files on other media, but a CD or DVD is usually where they are found.

between the operands. The first name is a path name and the second name is a particular file name. That is, the second name is *relative* to the path specified by the first name.<sup>35</sup>

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

Current releases of zPDT have expanded the possible formats to include these:

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

The first example follows the original requirements. The second example uses a single fully qualified name. The third example causes the specified file name (my.tdf) to be relative to the directory used to start zPDT operation. The last example depends on the keyword TAPES to indicate that data files are relative to the directory above TAPES.<sup>36</sup>

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

### 3.3.12 The awsctc device manager

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

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

Multiple devices may be defined for this device manager. Chapter 16, “Channel-to-channel” on page 299 describes the setup and usage of this device manager.

<sup>35</sup> 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.

<sup>36</sup> This convention was used in the original OMA support and is documented in IBM publication SC53-1200.



## zPDT commands

zPDT commands are entered as normal Linux line commands in a Linux terminal window. If zPDT is running (that is, the System z function is operating) then zPDT commands directed to the System z must be entered in a Linux window that is owned by the Linux userid that started the System z function.<sup>1</sup> This is not normally an issue unless multiple zPDT instances are running, each under a separate Linux userid.

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

The commands described in this chapter are the “native” zPDT commands. The Information Technology Company (commonly known as ITC) offers a completely packaged zPDT system as an ITC product known as uPDT. All the commands described in this chapter also apply to uPDT systems; in addition, ITC also provides a graphics interface (GUI) to automate common startup, shutdown, backup, restore, and other important processes. The GUI operation is not described in this document; more information is available from ITC.<sup>2</sup>

The IBM zD&T product also offers additional commands and utilities for zPDT and these are described in zD&T documentation.

<sup>1</sup> This means the same Linux user who issued the **awsstart** command must enter any additional zPDT commands that affect that instance of System z operation.

<sup>2</sup> Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 or [sales@itconline.com](mailto:sales@itconline.com)

## 4.1 The commands with examples

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

Most of the commands have a *help* option, usually invoked by an **-h** operand.<sup>3</sup> 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 with a Linux **man** command using the zPDT command name as the operand, as in the following example:

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

The dollar sign (\$) or pound sign (#) in examples (and in many examples throughout this book) represent the Linux prompts.

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

### 4.1.1 The adstop command

[The adstop command is no longer available.]

### 4.1.2 The alcckd command

The **alcckd** command creates (and formats) a Linux file that may be used as an emulated 3380 or 3390 DASD unit. The file is formatted to correspond to 3380 or 3390 tracks and cylinders in CKD format, but is otherwise not initialized. A utility program (such as ICKDSF) must later be used to create a volume label, VTOC, and so forth. A standard model (3380-1, 3380-2, 3380-3 or 3390-1, 3390-2, 3390-3, 3390-9) may be specified to establish the size of the emulated device, or a “non-standard” model number may be used<sup>4</sup>, or a specific number of cylinders may be specified. zPDT need not be operational when using this command. (zPDT might need to be restarted, with an updated devmap, to use the newly created CKD device.)

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

Where:

file-name is a Linux file name.

<sup>3</sup> In some cases, a question mark (?) operand can be used in addition to the -h operand.

<sup>4</sup> The non-standard sizes are intended only for 3390 devices.

`-ddvice-type` is a device type, optionally with a model number. The following list presents the standard 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 may 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  $20 \times 1113 = 22260$  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, or 268,435,456 cylinders for an extended address volume 3390.

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

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

`-rf` performs the `-rs` function and reinitializes any emulated tracks with incorrect formats; the 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 and are described in Chapter 13, “Additional zPDT notes” on page 253.

`-f4` causes the new volume to be formatted in 4K blocks. Some Linux distributions (for System z) cannot format a CKD volume and this option is for use with those distributions.

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

If more than 65520 cylinders (for a 3390) are specified, an extended address volume (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 already exists.
16	File not found or file name is invalid.
17	Drive not ready.
19	Disk not valid.
20	Not an emulated CKD volume.
21	Emulated CKD format is not valid

Examples of command use:

\$ <code>al cckd /z/WORK01 -d3390-3</code>	(create new emulated 3390-3 volume)
\$ <code>al cckd /tmp/222222 -d3390 -s100</code>	(create small 3390 volume, 100 cylinders)

```
$ alckd /z/WORK01 -rs
```

(verify format of CKD volume)

### 4.1.3 The **alcfba** command

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

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

Where:

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

-ddevice-type:

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

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

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

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

-c change the volser of an existing FBA volume.

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

-r display the attributes of an existing FBA volume.

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

Return values are 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 already exists.                          |
| 16     | File not found or the file name is not valid.           |
| 17     | Drive not ready.  |
| 19, 20 | Disk not valid.   |

Command examples are as follows:

```
$ alcfba /z/TEMP01 -d9336-1 -vSCRTCH
```

```
$ a1cfba /tmp/444444 -d9336 -s2000B -vMYVOL1
$ a1cfba /z/TEMP01 -c -vWORK99
```

#### 4.1.4 The **ap\_create** command

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

```
ap_create -a n
```

Where:

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

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

#### 4.1.5 The **ap\_destroy** command

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

```
ap_destroy -a n
```

Where:

n is the number of a defined cryptographic coprocessor.

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

#### 4.1.6 The **ap\_query** command

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

```
ap_query
ap_query -a n
```

Where:

n is the number of a defined cryptographic coprocessor.

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

#### 4.1.7 The **ap\_von** and **ap\_voff** commands

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

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

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

Where:

*n* is the number of a cryptographic coprocessor.

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

Emulated cryptographic coprocessors defined in the devmap are automatically made online when zPDT is started. The **ap\_von** and **ap\_voff** commands are not normally used, although they become relevant when **ap\_create** or **ap\_destroy** commands are used.

#### 4.1.8 The **ap\_vpd** command

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

```
ap_vpd -a n
```

Where:

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

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

#### 4.1.9 The **ap\_zeroize** command

The **ap\_zeroize** command erases (zeros) the content of a specified emulated cryptographic coprocessor, or a subset of a coprocessor. zPDT must be running when this command is used.

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

Where:

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

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

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

When a new cryptographic coprocessor is used (or when one is zeroized), it must be reinitialized. This is normally done with the ICSF utility, as briefly explained in Chapter 17, “Cryptographic usage” on page 307.

#### 4.1.10 The **attn** command

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

```
attn device-number
```

Where:

*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 zPDT operation this command is probably not used.

A command example is as follows:

```
$ attn 590
```

#### 4.1.11 The **aws\_bashrc** and **aws\_sysctl** commands

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

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

The **aws\_sysctl** command makes required modifications to `/etc/sysctl.conf` and then executes `/sbin/sysctl`. The **aws\_sysctl** command must be executed with *root* authority. The statements this command adds to `/etc/sysctl.conf` are appropriate for many zPDT users, but might need to be manually modified for larger zPDT instances. This is discussed further in Chapter 5, “zPDT installation” on page 95

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

#### 4.1.12 The **aws\_findlinuxtape** command

This command lists any SCSI tape drives that are connected to the underlying PC:

```
aws_findlinuxtape
```

This command lists both the *st* and *sg* numbers currently associated with each tape drive found. See 14.2.1, “The awsscsi device manager” on page 283 for more information about the device identifiers used with SCSI tape drives. zPDT need not be operational when using this command. Remember that the *sg* number can change when Linux is booted.

#### 4.1.13 The **aws\_tapeInit** command

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

```
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 six characters, letters and/or numbers with no special characters. (This is slightly more restrictive than volume serial numbers created by other means.) The *volser* is automatically translated to upper-case EBCDIC. zPDT need not be operational when using this command.

Command examples are:

```
$ aws_tapeInit 222222 /z/tapexxyyzz      (creates volser 222222)
```

```
$ aws_tapeInit tape01 /tmp/mywork/TAPE01    (creates volser TAPE01)
```

#### 4.1.14 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. zPDT need not be operational when using this command.

```
aws_tapeInsp file-name
```

The *file-name* is the Linux file name of an emulated tape volume. Simple tests are made to determine if the file is in awstape format. If it appears to be in awstape format it is checked for VOL1, HDR1, and HDR2 labels. The dataset name and 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.

A command example is:

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

#### 4.1.15 The awsckmap command

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

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

Where:

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

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

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

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

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

--dev lists detailed device information from the devmap.

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

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

#### 4.1.16 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.) zPDT must be operational when using this command.

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

```
{ [dev-address] -a }
```

Where:

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

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

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

The text operand is normally included in single 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'
```

#### 4.1.17 The awsmount command

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

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

Where:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
For tape drives (emulated or SCSI)
$ awsmount 580 -q          query currently mounted file
$ awsmount 580 -m /tmp/tapevol/123456  mount emulated volume
$ awsmount 580 -o /z/654321          replace mounted volume
$ awsmount 580 -u          unmount current volume
$ awsmount 580 -x          (or --run) unmount current volume
$ awsmount 580 -b          backspace over tape mark
$ awsmount 580 -f          forward space over tape mark
$ awsmount 580 -s          rewind tape volume
$ awsmount 580 -t          write tape mark
$ awsmount 580 -c /tmp/mytape1       mount and use compression

For OMA tapes (using device number 180 as an example)
$ awsmount 180 -q          query currently mounted file
$ awsmount 180 -m /tmp/oma/11111     mount emulated volume
$ awsmount 180 -o /z/oma/dosvol      replace mounted volume
$ awsmount 180 -u          unmount current volume
$ awsmount 180 -x          (or --run) unmount current volume
$ awsmount 180 -b          backspace over tape mark
$ awsmount 180 -f          forward space over tape mark
$ awsmount 180 -s          rewind tape volume

Disks and printers (using 300 and 00E device numbers)
$ awsmount 300 -q          query mounted file name
$ awsmount 300 -m /z/LOCAL1          mount emulated volume
$ awsmount 00E -m /tmp/print1         printer output file
$ awsmount 300 -o /z/LOCAL2          replace mounted volume
$ awsmount 300 -u          unmount current volume

aws3270 (local 3270 sessions; device number 702 for example)
$ awsmount 702 -q          query tn3270 client
$ awsmount 702 -d          force a disconnect

awsscsi (connect SCSI tape drives, using device number 580 for example)
$ awsmount 580 -m /dev/sg3          connect SCSI tape drive
```

### 4.1.18 The **awsstart** command

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

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

Where:

- noosa creates a zPDT environment without any OSA components.
- map is optional before the devmap file name.
- file-name* is the name of a device map file.
- clean causes all previous logs and traces to be deleted.
- localtoken causes a local USB token to be used, ignoring any remote license servers.

Use of the --noosa parameter is unusual and should be done only at IBM direction. zPDT maintains a variety of logs and traces in the ~/z1090/logs directory. Note that this is a subdirectory of the userid that starts zPDT. The contents of the logs directory can grow over time. If no zPDT problems are under investigation, using the --clean parameter ensures that only currently relevant logs and traces (from the zPDT instance just being started) will appear in the directory.<sup>5</sup>

A zPDT system may be configured to use a remote license manager. If the owner wants to temporarily use a local token without changing the remote license manager configuration details, the --localtoken option may be used. The details may be found in “zPDT licenses” on page 147.

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

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

#### 4.1.19 The awsstat command

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

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

Where:

- i *n-seconds* indicates the list should be repeated every n-seconds. If the n-seconds parameter is not provided, the default is 400 seconds.
- device-list* is list of device numbers. If no device-list is provided, all defined emulated devices are listed. A range of device numbers may be specified, or the name of a device manager.
- s or --sort may be used to sort the awsstat output according to the indicated criteria. By default it is listed in subchannel order.

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

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

<sup>5</sup> zPDT automatically manages the logs; use of the --clean option is not normally necessary.

An example of the command and 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<sup>6</sup> commands issued to the device since the last IPL. The 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 owned by this device manager)
```

#### 4.1.20 The awsstop command

The **awsstop** command ends zPDT operation. This operation ends abruptly, with no warning to the System z operating system. You would normally shut down your System z operating system in a orderly manner before using **awsstop**.

```
awsstop
```

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

```
$ awsstop
```

#### 4.1.21 The card2tape command

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

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

Where:

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

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

The compression option saves space in the emulated output file, but is not compatible with other platforms that can use awstape files. It does not indicate the use of hardware tape compaction, such as IDRC. The output is in unblocked 80-byte records, 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 translation are the same as used for the awsrdr device manager. The -a or --ascii parameters may be used to force translation. The EBCDIC/ASCII translation table used cannot be changed.

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

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

<sup>6</sup> SSCH is the "Start Subchannel" instruction for System z. Counting I/O operations can be a fuzzy topic due to command chaining in channel programs or queued I/O devices.

### 4.1.22 The **card2txt** command

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

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

Where:

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

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

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

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

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

### 4.1.23 The **ckdPrint** command

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

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

Where:

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

The program prompts for the range of tracks to dump. These are entered as four decimal numbers separated by blanks. The numbers, 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, use Ctrl+C. 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 as follows:

```
$ 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
```

### 4.1.24 The **clientconfig** command

The **clientconfig** command provides an interactive menu function to assist in configuring a remote license server and a Unique Identity Manager (UIM) that provides consistent System z serial numbers. This command must be run as *root*.<sup>7</sup> zPDT need not (and probably would not) be operational when using this command.

```
clientconfig
```

A description of the command use is found in “zPDT licenses” on page 147.

### 4.1.25 The **clientconfig\_authority** command

The **clientconfig\_authority** command adds a Linux userid or removes a Linux userid from a list of userids that may issue the **clientconfig** command. Normal use of **clientconfig** requires the user to operate as *root*. The **clientconfig\_authority** command allows the installation to avoid use of *root* when changing license server configurations. The **clientconfig\_authority** command itself must be run as *root*,<sup>8</sup> but it is used only once for a given userid. zPDT need not be operational when using this command.

Once a userid is authorized with this command, he would use the Linux **sudo** command to execute **clientconfig**.

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

-a adds the indicated userid to the list of userids that are allowed to issue the **clientconfig** command.

-d removes the indicated userid from this list.

A command example is as follows:

```
# clientconfig_authority -a ibmsys1
```

### 4.1.26 The **clientconfig\_cli** command

This command provides a Linux command-line interface (non-interactive) for the **clientconfig**. The command-line interface could be used in a Linux script, if desired. You must be *root* to use this command.<sup>9</sup> zPDT need not be operational when using this command. The syntax is as follows:

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

Where:

-g1s1 specifies a Gen1 license server, using a numeric IP address or a domain name.

-g1s2 specifies a backup Gen1 license server, using a numeric IP address or a domain name.

-g2s1 specifies a Gen2 license server, using a numeric IP address or a domain name.

-g2s2 specifies a backup Gen1 license server, using 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 request a display of the currently configured license servers.

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

<sup>7</sup> The zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

<sup>8</sup> The zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

<sup>9</sup> The zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

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

Command examples are:

```
# clientconfig_cli -gls1 my.remote.licenses.net
# clientconfig_cli -g2s1 192.168.1.107 -g2s2 123.321.111
# clientconfig_cli -l
```

### 4.1.27 The **cpu** command

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

**cpu** cp-address

Where:

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

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

\$ <b>cpu</b> 1	(select second CP as the default CP, which is CP number 1)
\$ <b>stop</b>	(place default CP in stopped state)
\$ <b>d psw</b>	(display PSW of the default CP)
\$ <b>start</b>	(start the default CP again)

### 4.1.28 The **d** command

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

```
d {r                                     }
  {p | psw                             }
  {pfx                                 }
  {g | gn                              }
  {y | yn                              }
  {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) }
```

```

{1so
{hn
}
```

Where:

*r* displays the current architecture mode.

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

*pf<sub>x</sub>* 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 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 use.

*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) may be used. The length is in hexadecimal.

*vp* displays primary virtual memory.

*vs* displays secondary virtual memory.

*vh* displays the home address space virtual memory.

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

*vr* displays one or more vector registers (z13 and later).

*access-reg* is the number of an access register.

*t* (just before a real address) indicates both hex and character displays are wanted.

A memory address not prefixed with *vp*, *vs*, *vh*, or *va* displays data at the System z real memory address.<sup>10</sup> Memory is displayed on 32-byte boundaries. If the specified address is not on a 32-byte boundary, the next lowest 32-byte boundary is used. Each memory line displayed ends with the protect key for that memory. As a general statement, the CP should be in a stopped state before any of these display functions are used.

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

A virtual address is meaningful only if an address space is active at the instant of the display.<sup>11</sup> When z/OS is in a wait state there may be no active address spaces. *As a general statement, these commands are not useful for application programming debugging unless there is a way to stop the CP while the application is actively being executed, with the appropriate virtual memory translation tables active.*

The leap second information block is described in Chapter 20, "Server Time Protocol (STP)" on page 337.

<sup>10</sup> 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.

<sup>11</sup> That is, if the appropriate segment tables (or region tables) for the target address space are indicated by the appropriate control registers for the default CP.

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 specified.
```

Examples of use are as follows:

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

### 4.1.29 The display\_gen2\_acclog command

The **display\_gen2\_acclog** command displays the access log maintained internally by a Gen2 license server. The displayed data is sent to Linux stdout and can be redirected to a normal Linux file if desired. 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; it cannot be used on a Gen2 client system.

**display\_gen2\_acclog**

There are no operands

### 4.1.30 The fbaPrint command

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

**fbaPrint** emulation-file-name

Where:

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

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

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

```
$ fbaPrint /z/VSE123
0 1                               (Dump two blocks)
```

### 4.1.31 The find\_io command

The **find\_io** command is used to identify potential OSA ports.

```
$ find_io
FIND_IO for "ibmsys1@linux-8jfl"
Path      Interface      Current      MAC          IPv4          IPv6
          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  *          *
...

```

Path	Interface Name	Current Settings						MTU**
		RxChkSum	TSO	GSO	GRO	LRO	RX VLAN	
F0	eth0	On*	On*	On*	On*	Off	On*	1500
F8	wlan0	Off	Off	On*	On*	Off	Off	1500

\* Enabling these functions may lead to poor zPdt Performance, please refer to your 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

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

The other data shown (State, MAC address, IPV4, and IPV6 addresses) is informational only. The IP addresses apply only to the base Linux. z/OS (or another System z operating system) could also address the interfaces with completely different IP addresses. All LAN interfaces known to Linux are shown; some of these may not be relevant or tested for zPDT usage. The MAC addresses for *tap* devices are artificial.

The **ethtool** parameters shown (TSO, GSO, GRO, LRO, RX VLAN) can affect performance. The MTU<sup>12</sup> size shown is the Linux (not z/OS) MTU, but it should be coordinated with the z/OS (or z/VM or z/VSE) MTU. Chapter 7, “LANs” on page 115 goes into more detail about these parameters.

### 4.1.32 The gen2\_init command

This command installs (or removes) the Gen2 license interface modules. The command must be run with *root* authority:<sup>13</sup>

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

Earlier versions of this command can trigger a web search for certain 32-bit modules. The license interface modules supplied with zPDT GA9 are 64-bit modules and no search for them is needed.

### 4.1.33 The hckd2ckd and hfba2fba commands

These commands deal with the migration of DASD volumes from a remote system to a zPDT system. In this context “migration” simply means “copy.” The are two client commands used with the migration utilities are:

<sup>12</sup> MTU is Maximum Transmission Unit, which is the number of bytes sent as a single unit over the LAN.

<sup>13</sup> The zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

<b>hckd2ckd</b>	Used with both z/OS and z/VM to migrate a CKD DASD volume.
<b>hfba2fba</b>	Used only with z/VM to migrate an FBA DASD volume.

zPDT need not be operational when using these commands. The general syntax of the client commands (entered on the Linux client machine, using a normal Linux command window) is as follows:

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

**Where:**

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

*:port* is a TCP/IP port number to be used by both the client and server program. It defaults to 3990. **Note:** recent experience indicates that the port number does not always default correctly; we suggest you always include the port number in the command.

*outfile* is a file name (on the current Linux system) where the migrated volume is to be placed (in awscd or awsfba format).

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

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

Either the *-u* or *-v* parameter must be supplied, but not both. These commands are described in Chapter 15, “DASD volume migration” on page 289.

Examples of commands that could 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
```

## 4.1.34 The interrupt command

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

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

**Where:**

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

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

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

Examples of use are as follows:

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

### 4.1.35 The `ipl` command

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

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

Where:

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

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

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

*gprparm* provides a string of characters that are inserted into the general purpose registers (as EBCDIC characters and using 32-bit registers), starting with register 0, placing four 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 may be set by the `cpu` command. The use of a *parm-value* completely depends on the operating system being used, and how that operating system is configured. As a general statement, it is not necessary to `clear` memory before loading an operating system.

The device indicated by the device-number must have *IPL text* installed; this installation 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 IPLed program tries to interact with a console. This could be an emulated 3270 session or the HMC console (also known as the “hardware console”). Messages sent to the HMC console appear in the Linux command window that issued the `awsstart` command to start zPDT<sup>14</sup>. Such messages (in the Linux command window) with the prefix “OPRMSG” and text may be sent to the HMC console with the `oprmsg` 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)
```

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
```

### 4.1.36 The `ipl_dvd` command

The `ipl_dvd` command emulates IPLing a DVD from the Hardware Management Console (HMC) on a larger System z. The DVD must contain files in a specific format for this function

<sup>14</sup> See “zPDT log files” on page 258 for additional ways to handle these messages.

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 System z distributions, a form of z/VSE distribution, and the COD z/OS system. zPDT must be operational for this command to be used.

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

Where:

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

-q causes the command to run in quiet mode.

-c (or --console) specifies the address (device number) of a local 3270 (in the active devmap). This 3270 is then used as an HMC 3270 session. *(At the time of writing, this function was not working with z/VM 6.2 and later. Instead, the int3270port function in the [system] stanza of the devmap can be used.)* The use 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 System z processor (normally the one listed first in the *processor* statement in the devmap) must be a standard cp. It cannot be an IFL. (This is a departure from standard System z 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.

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 .ins file.
```

An example of use is as follows:

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

### 4.1.37 The ldk\_server\_config command

This command may be used if a remote Gen2 license server is used to provide licenses to your client. It is a command-line alternative to the interactive **clientconfig** command. You must be *root* to use this command.<sup>15</sup> zPDT need not (and probably is not) active when this command is used. The syntax is:

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

Where:

*server.url.address* is the address of your Gen2 server. The address may be a URL (domain name) or a numeric IP address. Multiple addresses, separated by spaces, may be listed if you have alternate Gen2 servers.

-d indicates that the currently active Gen2 server addresses are to be listed.

-L indicates that no remote Gen2 server is to be used.

<sup>15</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

Examples are:

```
# 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 may be done with the `clientconfig_cli` command.

### 4.1.38 The listVtoc command

The `listVtoc` command provides a detailed listing of the VTOC of a z/OS CKD volume. It assumes the emulated CKD volume has been initialized with a label and a z/OS compatible VTOC. This command may be used while zPDT is operational, but it would normally be used when zPDT is not operational. The syntax is as follows:

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

Where:

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

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

Examples of use are as follows:

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

### 4.1.39 The loadparm command

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

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

Where:

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

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

This value set by this command is available to the operating system during the next IPL. If an IPL parameter is provided as part of an `ipl` command, it overrides any existing `loadparm` value and is stored as the current *value*. A parameter set this way is maintained only during zPDT operation; it is not retained across multiple zPDT startups. This command is probably never used by normal zPDT customers.

Return values are as follows:

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

Examples of command usage are as follows:

```
$ loadparm 0A8200P
```

```
$ loadparm -d
```

#### 4.1.40 The **managelogs** command

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

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

Where:

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

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

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

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

#### 4.1.41 The **memld** command

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

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

Where:

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

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

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

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')
```

#### 4.1.42 The **mount\_dvd** command

The **mount\_dvd** command identifies the Linux mount point for a DVD (or CD) that is to be processed as if it were mounted in the DVD drive of a System z HMC. zPDT must be operational when using this command.

```
mount_dvd complete-path
```

Where:

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

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

An example of the command is as follows:

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

#### 4.1.43 The `msgInfo` command

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

```
msgInfo message-number
```

Where:

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

No return codes are defined for this command. An example of usage is 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 needed but
AWSINF013I     if errors are present the DEVMAP cannot be used to start system.
```

All message numbers are in the form `AWScnnns`, where:

`ccc` is the component code issuing the message.  
`nnn` is the message number within the component.  
`s` is the message severity (**D**ebug, **I**nformation, **W**arning, **E**rror, **S**evere, **T**erminal)

The message code specified on the **msgInfo** command can omit the AWS prefix and the severity code. For example, **msgInfo chk082** is sufficient. There is also an environment variable named `Z1090_MSG` to control message formatting.<sup>16</sup> It may be set to `FULL` (the default), `CODE` (which will only print the message number and no text), `TEXT` (which prints the message text and no code) and `SHORT` (which drops the AWS prefix on the message number).

#### 4.1.44 The `oprmsg` command

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

```
oprmsg {text}
```

Where:

---

<sup>16</sup> This environmental variable can be set with an **export** statement in the Linux shell.

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

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

The **oprmsg** command is entered in a Linux command window and processed by the Linux shell executing that window. This means 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 we suggest inclosing the operand in single quotes, as shown in the examples below.

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 use are as follows:

```
$ oprmsg 'V CN(*),ACTIVATE'
$ oprmsg 'V 700,CONSOLE'
$ oprmsg 'D A,L'
$ oprmsg m '* "hello Colin" ' (result: m * 'hello Colin ' )
```

## 4.1.45 The pdsUtil command

The **pdsUtil** command is a Linux command that reads (or rewrites) members of a z/OS partitioned data set. z/OS is normally not operational when this command is used. The target data set must be a PDS (not PDSE) with FB records. This command cannot change the length or number of records in the PDS member. Record length is not limited to 80 bytes. The general operation is to *extract* the PDS member (to Linux), *edit* the Linux file, and then *overlay* the original PDS member with the changed data. Automatic ASCII/EBCDIC translation is provided.

The syntax is 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>]
```

Where:

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

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

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

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

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

The PDS member name may be specified in any one of three ways. Using parenthesis around the member name requires that the parenthesis be escaped (so that the Linux shell 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 may 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 dataset. Also, **pdsUtil** operates in your Linux system and has no knowledge of any RACF protection that might apply to the target PDS.

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

## 4.1.46 The query command

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

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

Where:

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

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

The return values are as follows:

```
0      Query 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.

#### 4.1.47 The `query_license` command

The `query_license` command displays the current zPDT license status for Gen2 licenses,<sup>17</sup> including the identity of the remote license server. zPDT need not be operational to use this command, but the user must have configured a Gen2 client environment.

```
query_license
```

There are no operands. This command is not relevant for local token usage or for a remote Gen1 license server. This command may be used on both Gen2 clients and Gen2 servers. For a client, only information relevant to that client is displayed. For a server, more general information is displayed; the display is described in “Managing the Gen2 server” on page 157.

#### 4.1.48 The `rassummary` command

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

```
rassummary [-s] [-t] [-d directory-name] [-c comp-name] [-u subcomp-name]
           [-b begin-time] [-e end-time] [-r rec-type]
```

Where:

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

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

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

`-c comp-name` indicates only records about the indicated component are to be displayed. This option is intended only for IBM internal use and is not further documented.

`-u subcomp-name` indicates only records about the indicated subcomponent are to be displayed. This options is intended only for IBM internal use and is not further documented.

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

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

`-r rec-type` indicates that only the specified record type is to be displayed. Valid types are TRACE, LOG, LOG\_REGBUF, QD\_DUMP, LOG\_EVENT, LOG\_APPEND, and QUICK\_DUMP. Multiple operands may be separated with a comma.

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

---

<sup>17</sup> See Chapter 8., “zPDT licenses” on page 147 for information about Gen2 licenses.

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

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

#### 4.1.49 The ready command

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

```
ready device-number
```

Where:

*device-number* is the "address" assigned to the emulated device in the devmap.

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

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

#### 4.1.50 The request\_license command

The **request\_license** command creates a file that you send to the IBM-authorized facility that generates Gen2 license files. This command is run with *root* authority.

```
request_license
```

More information is in "Gen2 License server" on page 156.

#### 4.1.51 The restart command

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

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

Where:

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

This command is seldom used. In some cases it may be used to assist an operating system that is stuck in an unusual situation such as a restartable disabled wait. It is also used to dump a z/VM system and to 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                (restart default CP, as set by the cpu command)
$ restart 2
```

### 4.1.52 The **scsi2tape** command

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

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

Where:

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

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

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

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

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

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

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

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

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

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
```

### 4.1.53 The SecureUpdateUtility command

**Important:** This command is replaced with the **Z1090\_token\_update** or **Z1091\_token\_update** command for zPDT release GA5 and later. Continue using **SecureUpdateUtility** for zPDT releases prior to GA5.

The **SecureUpdateUtility** command is used to manage zPDT license lease dates<sup>18</sup> in the zPDT token. zPDT must not be running when this command is used. (Note that Linux warning messages are issued a month before the lease date in the token expires.) You must have *root* authority<sup>19</sup> and be in the `/usr/z1090/bin` directory before issuing this command.<sup>20</sup>

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

The first form (`-r`) writes a *request* (`.req`) file for the token currently connected to the computer. Only one token should be connected when using this command. This request file is unique to the token currently connected. Note that some users (typically with 1091 tokens) do not require creation of a request file. Your zPDT supplier will determine if a request file is needed.

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

The request file is sent to a license processing facility that uses it to create the update file. The update file is then sent to the user who applies it with the **SecureUpdateUtility** command. An update file is unique to a token number and may be used only once. The license processing facility may return a `.upw` file, a `.zip` file, or both. See “Gen1 token activation and renewal” on page 167 for information about handling a `.zip` token file. You must have a `.upw` file in order 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 myreq.req file in Linux)
                               (Send the req file for processing; receive an upw file in return)
# SecureUpdateUtility -u myreq.upw (apply the update file)
# exit                       (exit from root)
```

### 4.1.54 The SecureUpdate\_authority command

The **SecureUpdate\_authority** command adds a Linux userid to, or removes a Linux userid from, a set of userids that may issue the **zpdtsSecureUpdate** command, which executes the **SecureUpdateUtility**, **Z1090\_token\_update**, or **Z1091\_token\_update** command internally, without requiring the user to operate as *root* and be positioned in the `/usr/z1090/bin` directory. The **SecureUpdate\_authority** and **zpdtsSecureUpdate** commands allow the installation to avoid usage of *root* when updating token licenses. The **SecureUpdate\_authority** command must be run as *root*,<sup>21</sup> but it is typically used only once for a given userid.

<sup>18</sup> Also known as the license expiration dates.

<sup>19</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with `./` (“dot slash”) to execute it from `/usr/z1090/bin`.

<sup>20</sup> The use of *root* may be avoided by using the **SecureUpdate\_authority** and **zpdtsSecureUpdate** commands.

<sup>21</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with `./` (“dot slash”) to execute it from `/usr/z1090/bin`.

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

- a adds the indicated userid to the list of userids that are allowed to issue the SecureUpdateUtility command.
- d removes the indicated userid 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 to permit token administration without requiring *root* authority. See 13.15, “Security exposures” on page 262 for details.

### 4.1.55 The `senderrdata` command

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

**senderrdata**

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

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

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

### 4.1.56 The `serverconfig` command

The `serverconfig` command works with an Gen2 server, which must be installed before this command is used:

```
serverconfig [ ]                (no operand)  
              [-u | --update]  
              [-r | --rules]
```

When the command is used with no operand, it displays a small interactive menu, described in “Gen2 License server” on page 156, 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 used by the LDK server. The `--update` option causes the server to reread the security file. The same functions performed by this command can also be performed 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.

#### 4.1.57 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.<sup>22</sup>

```
serverconfig_cli [-a[y|n]] [-l[y|n]] [-u]
```

The `-a` option enables or disables the Gen2 license server, without completely stopping it. All connected zPDT systems will 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. The functions are explained in more detail in “Gen2 License server” on page 156.

Example:

```
# serverconfig_cli -ay -u
```

This example enables the Gen2 license server (which we assume was previously disabled) and rereads the security rules.

#### 4.1.58 The `settod` command

The `settod` command sets the specified time/date in the System z Time Of Day (TOD) clock during the next IPL of an active zPDT system. The TOD change is not carried across restarts of zPDT. When used, this command is normally issued after the `awsstart` command and before an `ipl` command. zPDT normally sets the emulated System z TOD clock to match the underlying PC TOD clock;<sup>23</sup> this command alters that normal action. A `settod` command issued while a System z operating system is active has no immediate effect; it takes effect only during a subsequent `ipl` command.

The full syntax is as follows:

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

If both date and time are present, they must be separated with a dash (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 time stamps that are stored in a zPDT token. This command provides a way to test System z software at future times (or past times). After the subsequent `ipl`, the System z

<sup>22</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with `./` (“dot slash”) to execute it from `/usr/z1090/bin`.

<sup>23</sup> The leap seconds value can create a difference between the PC clock and the System z TOD clock.

TOD clock is incremented in the normal way, starting at the time/date specified in the **settod** command.

Assume the current date and time (in the PC hardware clock) is July 20, 2017 at 1 PM:

```
$ 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. However, we suggest you always enter a full date or time, or both.

## 4.1.59 The snapdump command

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

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

Where:

*comp-name* is a component name; only information related to this component is obtained. This option is intended only for IBM internal use and is not further documented.

*subcomp-name* is a subcomponent name; only information related to this subcomponent is obtained. This option is intended only for IBM internal use and is not further documented.

*desc-text* is descriptive text (in quotation marks).

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

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

An example of the command is as follows:

```
$ snapdump -d "This is a test"
```

## 4.1.60 The st command

The **st** (store) command is used to alter registers or memory in a CP (or zIIP or zAAP or IFL). zPDT must be operational when using this command. The syntax is similar to that for the **d** command.

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

```

{vrm xxx      (z13; m=0..31)      }
{lso (operands explained in "Server Time Protocol (STP)" on page 337)}
{hex-addr xxx      }

```

Where:

*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 may 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 complete.
-2     No arguments specified.

```

Examples of use are as follows:

```

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

```

### 4.1.61 The start command

The **start** command starts a CP that was in the stopped state (due to a prior **stop** command). zPDT must be operational when using this command.**start** [CP-number]

```

start [a11]
      [CP-number]

```

Where:

*CP-number* is the target CP number. If this operand is not specified, the CP number set by the **cpu** command is used. The constant **a11** may be used instead of a cpu number.

The return values are as follows:

```

0      Operation complete.
12     CP number is invalid.

```

16 Unable to initialize the manual operation interface.

Command examples are as follows:

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

### 4.1.62 The stop command

The **stop** command places a CP in the stopped state. It may be restarted with a **start** command or a reset function. zPDT must be operational when using this command.

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

Where:

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

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

The return values are as follows:

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

Command examples are as follows:

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

### 4.1.63 The storestatus command

The **storestatus** command causes certain CP registers to be stored in fixed memory locations, as defined in z/Architecture. zPDT must be operational when using this command. This command might be used before taking a stand-alone dump.<sup>24</sup>

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

Where:

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

The CP must be in the stopped state (via a **stop** command) when **storestatus** is issued. A **storestatus** is no longer required before taking a stand-alone dump of z/OS.

An example of the command is as follows:

```
$ stop all
$ storestatus
```

<sup>24</sup> Recent z/OS and System z usage have removed the need for this command. The System z (including zPDT) is primed by z/OS to perform a store status command before the next IPL.

#### 4.1.64 The storestop command

[The storestop command is no longer available.]

#### 4.1.65 The stpserverstart command

The **stpserverstart** command is used to start the Server Time Protocol (STP) function daemons. The STP function is typically used only for a basic sysplex configuration. The `/usr/z1090/bin/wnsCCT` 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*.

**stpserverstart**

This command also adds the STP function to the **cron** lists of the Linux system. This causes the STP function to be restarted (if it fails) and to be automatically started when the Linux system is rebooted. This command applies to all base systems (Linux systems) involved in the STP configuration. In an STP sense, some can be considered clients and others servers but this command is used with *all* Linux base systems involved.

The STP function is explained in more detail in Chapter 20, “Server Time Protocol (STP)” on page 337. You must configure the `/usr/z1090/bin/CCT_data` file before starting an STP server or client.

#### 4.1.66 The stpserverstop command

The **stpserverstop** command causes a running STP function to be stopped. The **cron** entries that automatically start the STP function are also removed. This command is not used as *root*.

**stpserverstop**

#### 4.1.67 The stpserverquery command

The **stpserverquery** command displays the status of any STP functions running on your system.

**stpserverquery**

#### 4.1.68 The sys\_reset command

The **sys\_reset** command performs a system reset (or system reset clear) function as defined by z/Architecture. zPDT must be operational when using this command.

**sys\_reset [normal | clear]**

Where:

*normal* is the default operation.

*clear* performs the additional architected clear function.

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

**\$ sys\_reset**

## 4.1.69 The **tape2file** command

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

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

Where:

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

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

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

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

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

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

## 4.1.70 The **tape2scsi** command

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

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

Where:

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

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

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

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

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

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

This command converts a logical tape volume to a real tape volume. The optional compression format used by zPDT awstape functions is recognized and processed, if present. The Linux name of the SCSI tape drive for this command is usually in the */dev/stn* group and not in the */dev/sgn* group. See Chapter 14, “Tape drives and tapes” on page 281 for more information about tape drive naming.

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
```

### 4.1.71 The tape2tape command

The **tape2tape** command copies a logical tape volume (in awstape format) to another logical tape volume (also in awstape format). Several optional operations may take place during the copy, including compressing or uncompressing the data. The primary purpose of the command is to compress or uncompress awstape files, or to summarize a file. zPDT does not need to be running to use this command.

```
tape2tape [-c      ] [-d      ] [-e nn  ] [-i      ]
          [--compress] [--dynainfo] [--eof nn] [--info  ]
                                     [-n      ]
                                     [--noinfo]

          [-s      ] in-file out-file
          [--scan]
```

Where:

- c or *--compress* causes the output tape to be compressed.
- d or *--dynainfo* displays tape content when each record is read. Otherwise, information is displayed only when a tape mark is encountered.
- e *nn* or *--eof nn* specifies the number of consecutive tape marks that indicate the logical end of the input file. The default is two tape marks.
- i or *--info* provides a summary of the tape volume. This is the default.
- n or *--noinfo* indicates no summary is to be displayed.
- s or *--scan* scans the tape, but no output is produced.
- in-file* is the name of a Linux file in awstape format.
- out-file* is the name of a Linux file that will be in awstape format.

The input file may be in the zPDT compressed format; this is handled automatically. The output file is compressed only if that option is selected. Both input and output files are in awstape format. This command cannot convert other Linux files to awstape format.

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

```
$ tape2tape -e 1 /tmp/111111 /z/222222.awstape
```

### 4.1.72 The tapeCheck command

The **tapeCheck** command verifies the internal format of a logical tape volume in awstape format. That is, it verifies that the awstape control bytes within the file are logically correct. zPDT does not need to be running to use this command.

```
tapeCheck file-name
```

Where:

*file-name* is a Linux file in awstape format.

This command is used to inspect awstape files that may have been corrupted. It could be used to check awstape files generated on another platform, to ensure they are compatible with zPDT. Note that some older platforms do not create correct awstape bytes at the end of a logical tape volume.

The return value is equal to the number of errors found in the awstape format. An example of the command is as follows:

```
$ tapeCheck /tmp/222222.awstape
```

### 4.1.73 The tapePrint command

The **tapePrint** command writes the content of an emulated tape volume (awstape format) to Linux stdout. zPDT does not need to be running to use this command.

```
tapePrint [-a      ] [-e      ] in-file  
          [--ascii] [--ebcdic]
```

Where:

*-a* or *--ascii* specifies that the emulated tape volume has ASCII characters.

*-e* or *--ebcdic* specifies that the emulated tape volume has EBCDIC characters. This is the default format.

*in-file* specifies a Linux file that is in awstape format.

Output is displayed block by block, in both hexadecimal and character format.

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

```
$ tapePrint /z/222222.awstape
```

### 4.1.74 The token command

The **token** command displays the characteristics of the zPDT token currently in use. This command should be used when zPDT is running.

```
token
```

There are no operands. Only the number of token licenses currently in use are displayed. That is, if the token allows three CPs, but only one CP is currently in use, then information for only one CP is displayed. An example of command use is as follows:

```
$ token  
CPU 0, zPDTA ... Serial 6186(0x182A) Lic=88570(0x159FA) EXP=4/15/2015 1090
```

The serial number is the effective System z serial number and may differ from the token serial number. The license serial number reflects the token serial number used to provide the zPDT license. The final output indicator is 1090 or 1091 where 1090 indicates the original zPDT version and 1091 indicates the appropriate version.

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.

### 4.1.75 The **txt2card** command

The **txt2card** command reads a Linux text file and creates a card image file (in EBCDIC).

```
txt2card in-file out-file
```

Where:

*in-file* is the name of a Linux text file (in ASCII). Each record must be 80 bytes or shorter.

*out-file* is the name of a Linux binary file that is written by this command.

Input records are extended (with blanks) to 80 bytes and then converted to EBCDIC. The ASCII/EBCDIC conversion table is fixed and cannot be customized.

There are no defined return values for this command. A command example is as follows:

```
$ txt2card /tmp/work2/config.txt /z/cards/deck1
```

### 4.1.76 The **uimcheck** command

The **uimcheck** command displays the state of the Unique Identity Manager (UIM) serial number (which is the System z serial number). zPDT need not be running when this command is issued; any user may issue the command.

```
uimcheck
```

There are no operands.

### 4.1.77 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 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*,<sup>25</sup> 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.

```
uimreset [-l] [-r]
```

-l indicates that the UIM serial number in the local UIM database should be erased.

-r indicates that the UIM serial number in both the local UIM database and the remote UIM server should be erased.

The Unique Identity Manager (UIM) function and its use are explained in detail in “zPDT licenses” on page 147.

### 4.1.78 The **uimserverstart** command

The **uimserverstart** command is used to start a remote UIM server. This command should always be issued by the same Linux userid because it saves a database in the home directory of that Linux user. It must not be started by *root*.

```
uimserverstart
```

---

<sup>25</sup> An alternative is to use **sudo**.

This command also adds the UIM server to the **cron** lists of the Linux system. This causes the UIM server to be restarted (if it fails) and to be automatically started when the Linux system is rebooted.

A UIM server is normally used only as part of a remote zPDT license server environment. It is not used when running a simple zPDT environment with a token connected to the local zPDT system.

The Unique Identity Manager (UIM) function and its usage are explained in detail in Chapter 8, “zPDT licenses” on page 147.

### 4.1.79 The **uimserverstop** command

The **uimserverstop** command causes a running UIM server to be stopped. The **cron** entries that automatically start the UIM server are also removed. This command is not used as *root*.

**uimserverstop**

### 4.1.80 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.<sup>26</sup>

**update\_license**

More information is in “Gen2 License server” on page 156.

### 4.1.81 The **Z1090\_AD\_CD\_install** and **Z1091\_AD\_CD\_install** commands

The command appropriate for your token (1090 or 1091) must be used. The discussion here is based on 1090 tokens. This command is used to install the IPL volumes for the AD-CD z/OS 2.1 (or later) release (and might apply to future z/VM, and z/VSE AD-CD releases). This command is used only for an IPL volume; other volumes are installed with the Linux **gunzip** command. This command is normally not run by *root*. zPDT cannot be running when this command is used; if it is running you will see messages about “No license available.”

You must have an appropriate AD-CD license in your token or license server for this command to work.

**Z1090\_AD\_CD\_install** infile outfile

Where:

*infile* is the distributed AD-CD volume that is the IPL volume. (For recent AD-CD z/OS systems, two volumes exist: the “normal” IPL volume and the recovery volume that is usually named SARES1.) These files (in the new format used with this command) have .zPDT as the file name suffix.

*outfile* is the installed volume, ready to use.

This command requires additional disk space in the directory containing the outfile. This additional space is the same size as the infile and is used for a temporary work file that is automatically deleted before the command completes. Be aware that the z/OS AD-CD IPL volumes (xxRES1 and SARES1) are prepared differently for 1090 ISV and zD&T systems. For example, a 1090 token cannot decrypt a SARES1 volume prepared for a zD&T customer.

<sup>26</sup> The zPDT program files are not normally in root’s PATH; you might need to prefix the command with ./ (“dot slash”) to execute it from /usr/z1090/bin.

Command examples are as follows:

```
Z1090_ADCD_install /tmp/C1RES1.zPDT /z/C1RES1
Z1090_ADCD_install /tmp/SARES1.zPDT /z/SARES1
```

#### 4.1.82 The Z1090\_token\_update and Z1091\_token\_update commands

**Important:** This command may not be relevant if your zPDT supplier provides token keys through a different method. Contact your zPDT supplier for details.

The command appropriate for your Gen1 token (1090 or 1091) must be used. The examples here are based on 1090 tokens. This command is new with the zPDT GA5 release and replaces the **SecureUpdateUtility** command.<sup>27</sup> You must be *root* and have */usr/z1090/bin* as the current directory<sup>28</sup> to use this command.<sup>29</sup> zPDT must not be operating when this command is used to update a token. The *-r* function may be used while zPDT is active.

```
Z1090_token_update [-r filename[.req]]          (or Z1091_token_update)
                  [-u filename.[zip | upw]]
                  [-status]
```

Where:

*-r* followed by an arbitrary file name creates a request file used to obtain token license updates.<sup>30</sup> The *.req* suffix is optional and will be automatically added if not present. When the request file is sent to the licensing facility, you should receive in return a *.zip* file, a *.upw* file, or both. See “Gen1 token activation and renewal” on page 167 for details about these files.

*-u* causes either the *.zip* or the *.upw* file to be installed. You could use the *.upw* file if you do not plan to install the AD-CD 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 associated with AD-CD usage are present.

This command obtains and installs updates for the zPDT license and for the licenses needed to install the AD-CD z/OS 2.1 (or later) system. (Future z/VM and z/VSE releases may also have similar requirements.) There is no harm in installing the token licenses needed for the AD-CD z/OS 2.1 (or later) release even if you have no immediate plans to use it. 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. Be patient.

Only one token should 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 myreq.req file in Linux)
                             (Receive an .upw or .zip file, or both)
# Z1090_token_update -u myreq.zip (apply the update file)
```

<sup>27</sup> You should continue to use the **SecureUpdateCommand** for zPDT releases prior to GA5.

<sup>28</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with *./* (“dot slash”) to execute it from */usr/z1090/bin*.

<sup>29</sup> The use of root may be avoided by using the **SecureUpdate\_authority** and **zpdtdSecureUpdate** commands.

<sup>30</sup> Some zPDT users (typically with 1091 tokens) might not require a *.req* file in order to receive token updates. Consult your zPDT supplier for details.

```
# exit
```

```
(exit from root)
```

### 4.1.83 The **Z1090\_removeall** command

The **Z1090\_removeall** command does exactly what the name implies. It removes zPDT (including the two token drivers) from a Linux system.

```
z1090_removeall
```

This command requires *root* authority and is appropriate only in unusual circumstances.<sup>31</sup>

### 4.1.84 The **z1090instcheck** command

The **z1090instcheck** command checks a number of installation criteria. It may be used whether or not zPDT is running. The same **z1090instcheck** command is used for zD&T systems.

```
z1090instcheck
```

There are no operands. This command is sensitive to the Linux distribution being used and to the level of that distribution. The output may vary with a new release of zPDT. The **z1090instcheck** command is used with both 1090 and 1091 systems.

Return values are 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
and so forth
```

Remember that the specific report changes with new zPDT releases and with the underlying Linux distribution. Some of the checks may produce warnings that you must evaluate for yourself. The help function (**z1090instcheck -h**) returns additional configuration suggestions.

### 4.1.85 The **z1090term** command

The **z1090term** command provides an ASCII terminal function that may be used to connect to the HMC-like ASCII terminal defined by the `intASCIIport` statement in a zPDT devmap. The syntax is as follows, where the `IPaddress` points to the base Linux system running zPDT and the port number is defined in the `intASCIIport` statement of the zPDT device map:

```
z1090term IPaddress:port
```

Examples are as follows:

```
$ z1090term my.remote.zpdt.com:7100
```

<sup>31</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with `./` ("dot slash") to execute it from `/usr/z1090/bin`.

```
$ 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; the executable file may be copied to other Linux systems where this terminal interface could be used to connect to zPDT.

#### 4.1.86 The **z1090ver** and **z1091ver** command

The **z1090ver** or **z1091ver** command displays the current zPDT version, with the date it was build. This information may be necessary when investigating a zPDT problem.

```
z1090ver                (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.

#### 4.1.87 The **zpdtsSecureUpdate** command

The **zpdtsSecureUpdate** command automatically switches to the `/usr/z1090/bin` directory, executes 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<sup>32</sup> or you must be enabled to execute this command via the **SecureUpdate\_authority** command.

```
# zpdtsSecureUpdate [-r | -u] Linux-file-name      (if you are root)
$ sudo zpdtsSecureUpdate [-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 is used in a manner that does not require root authority by the user. See 13.10, “Security exposures” on page 268 for details.

---

<sup>32</sup> The zPDT program files are not normally in root's PATH; you might need to prefix the command with `./` (“dot slash”) to execute it from `/usr/z1090/bin`.



## zPDT installation

**Important:** This chapter describes the basic zPDT installation process. Your product package might provide an enhanced installation method provided by your business partner or with the ZD&T product. If this is confusing, please discuss installation methods with your zPDT provider.

zPDT operates as a normal Linux application. We suggest that the general procedures described here be followed for initial zPDT installation and use. After you gain experience with zPDT, you might explore other installation and usage arrangements.

**Important:** For zPDT installation, *do not* work as *root* except when instructed to do so.

### Starting requirements

This document does not provide detailed ordering information for zPDT. The ordering process differs for various categories of users and for different countries. At a high level, you need to acquire the following:

- ▶ A PC Linux operating system that is supported by zPDT. This *base Linux* is not supplied by IBM. It must be ordered or downloaded from a web site.
- ▶ A zPDT token (which *might* need to be activated through an IBM business partner, zPDT supplier, or through IBM Resource Link®). An alternative to physical tokens exists and is discussed in Chapter 8, “zPDT licenses” on page 147.
- ▶ The zPDT software, which must be installed before the token can be activated (if it is not supplied already activated). The base zPDT software does not include any z System Operating Systems.
- ▶ Whatever z System software you plan to use, in a format usable with zPDT. This *might* require a different ordering process than ordering zPDT itself. Typically, this is an AD-CD package, as outlined in Chapter 6, “AD-CD installation” on page 105. Much of the material in this document assumes that you will install the z/OS AD-CD system. If you are installing different z System software, you must obtain specific instructions for zPDT from the supplier of that software.

## 5.1 Linux Installation

There are many Linux distributions, and each of these have multiple levels available. We cannot provide specific testing or instructions for using all the possible combinations. The following guidelines may be useful:

- ▶ zPDT does not care how you partition your disk(s), provided you have sufficient room for whatever z System volumes you need. A very simple disk plan might be this:
  - A 20 GB partition for all the Linux files and the zPDT program.
  - An appropriate swap partition, perhaps 10 GB
  - A large partition for emulated z System volumes, typically *at least* 200 GB.

The separate partition (and corresponding separate Linux file system) for the emulated z System volumes allows the base Linux to be replaced easily without destroying the emulated volumes.

- ▶ A given zPDT release is built with specific Linux distributions at specific levels. The distributions are RHEL, SLES<sup>1</sup>, and Ubuntu and the specific levels are described in “zPDT releases” on page 24. The levels of these distributions are current at the time the development of the zPDT release was started. If you elect to use different or newer Linux bases then *you* might need to resolve some library mismatches or other small problems.
- ▶ If you are using 1090 or 1091 tokens,<sup>2</sup> your base Linux must include a 32-bit version of the libstdc++ library. You might need to obtain and install the library yourself.<sup>3</sup> At the time of writing the required library was:
  - For Red Hat: libstdc++.i686                   (**yum install libstdc++.i686**)
  - For SUSE: libstdc++6-32bit               (**yast** or **zypper install libstdc++6-32bit**)
  - For Ubuntu: lib32stdc++6               (**apt-get install lib32stdc++6**)

The library names noted here are the “search names” that you might use to locate the library. Each library has a more complete name such as libstdc++6-32bit-7.1.1+r248970-1.4.x86\_64 for SUSE. The library need not be installed while you are doing the base Linux installation, but must be installed before zPDT can be installed.

- ▶ zPDT requires the *libncurses5* library. Some recent Linux distributions have the *libncurses6* library and this does not work for the current zPDT version. You need to install *libncurses5* or perhaps *ncurses-compat-libs*, using **yast**, **yum**, **zypper**, or **apt-get**, as appropriate.
- ▶ zPDT is not sensitive to the desktop manager. Some developers use GNOME and others use Xfce or KDE.
- ▶ You probably need the x3270 package. It is generally not included with current Linux distributions.
- ▶ We found that performing an online update for the Linux distribution is advisable.
- ▶ You must manage whatever firewall and other security functions that you install with your Linux. We suggest initially disabling any firewall when first working with zPDT. After you are familiar with zPDT operation, you can reestablish the firewall functions. If you have external TCP/IP connections (for local 3270 connections, for OSA connections, for license servers, for STP, or for CTC connections) you must provide appropriate port *holes* in any firewall you use.
- ▶ We found it advisable to select several additional packages during Linux installation:

<sup>1</sup> SLES is not an option for zD&T users.

<sup>2</sup> If you are using a software-only license server then this library is not required.

<sup>3</sup> The required library might be included with some SUSE releases, possibly by specifying 32-bit support.

- We suggest you install `smpppd`, if it is available and not installed by default. This is used for traditional network support and might be needed to install `x3270`.
- Newer `x3270` versions require `expect` to be installed.
- Install `vsftp` or `sftp` or some other Linux FTP package. It is not required for zPDT, but you might want to transfer files between Linux and z/OS using FTP.
- ▶ Include Perl; it is not required for basic zPDT operation, but is used by some RAS functions. Perl is typically included automatically in the Linux installation process.
- ▶ Select Universal Time (UTC) for your base PC, if possible. (This might not be possible if you also run Windows on the same PC.
- ▶ zPDT installation includes two Linux packages named `sntl-sud` and `shk-server`. You might find later versions on these packages elsewhere. *Do not install them*. Use only the versions supplied with zPDT.
- ▶ **Do not** use zPDT while working as *root*, except as instructed

## 5.2 The zPDT program package

The zPDT program package is typically obtained by a download. Two zPDT packages are available, one for ISVs with 1090 tokens and one for zD&T with 1091 tokens (or the equivalent Gen2 tokens).<sup>4</sup> The proper package must be used with the proper token.

For each package, two Gen1 token drivers, the zPDT modules, and the license agreement are all in a single (non-rpm and non-deb) file. The file contains Red Hat, SUSE, and Ubuntu versions of the zPDT code.<sup>5</sup> The proper version is automatically installed on your system.

## 5.3 zPDT installation

Decide on the Linux userid you want to use for zPDT. We use *ibmsys1* in all our examples, but you can select any userid that is eight characters or fewer (unless the *system\_name* statement is used in the devmap, in which case there is no special limit on the length of the Linux userid).

In the examples throughout this document, the dollar sign prompts (\$) indicate a non-root userid; the number sign prompts (#) indicate we are working as *root*. We suggest that you always log in as *ibmsys1*<sup>6</sup> and then use an `su` command to switch to *root* when needed.<sup>7</sup> The following directions assume that a single zPDT instance is used. (Multiple zPDT instances require multiple userids, such as *ibmsys2* and *ibmsys3*.)

If you have not already done so, create user *ibmsys1*. By default, userid *ibmsys1* has `/home/ibmsys1` as its home directory and some zPDT control files appear in subdirectories here. We created file system `/z` as a separate partition during our Linux installation.<sup>8</sup> We want userid *ibmsys1* to own this file system:

```
(log on as ibmuser1)
# su -                               (switch to root)
```

<sup>4</sup> There are alternatives to these tokens, but we use these token names for descriptive purposes.

<sup>5</sup> The three zPDT versions (in each package) are needed because the three Linux bases are typically at slightly different library levels.

<sup>6</sup> There is nothing special about userid *ibmsys1*. We consistently use it to illustrate zPDT operation.

<sup>7</sup> Another option is the use of the `sudo` facility.

<sup>8</sup> This might not be the case for the IBM Open Client, but we ignore this exception here.

```
# chown ibmsys1 /z (user ibmsys1 to own emulated volume directory)
```

A single executable file is used to install the zPDT software.<sup>9</sup> The file name changes with maintenance releases, but has the following general format:

```
z1090-1-10.55.05.x86_64 (verify your exact file name)
z1091-1-10.55.05.x86_64 (for zD&T)
```

The single file contains the following items:

- ▶ An `sntl-sud` rpm at the correct level (A driver for the Gen1 tokens)
- ▶ A `zpd-tshk-server` rpm at the correct level (Another token program)
- ▶ The primary z1090 rpm for SUSE Linux
- ▶ The primary z1090 or z1091 rpm for Red Hat Linux
- ▶ The primary z1090 or z1091 deb for Ubuntu
- ▶ An *installer program* that displays a license and then installs the rpms or debs. The correct rpm or deb (Red Hat or SUSE or Ubuntu) is automatically selected for your base Linux system.

Place the zPDT package file in a convenient library, such as `/tmp`.

```
(log in as ibmsys1)
$ su - (change to root)
# cd /tmp (if the file is in /tmp)
# chmod u+x z1090-1-10.55.05.x86_64 (make file executable)
```

Execute the file, which is the *installer program*:

```
# ./z1090-1-10.55.05.x86_64 (verify the exact file name first)10
```

Scroll through the license that is displayed and reply to the question at the end. The various components are then installed automatically. The zPDT installer program performs the following tasks, removing existing versions of these programs as needed:

- ▶ Two prerequisite token modules are installed.
- ▶ The z1090 or z1091 rpm or deb is installed, mostly in `/usr/z1090/bin`.
  - `/usr/z1090` is used for both ISV and zD&T versions; `/usr/z1091` is not used.
- ▶ A set of man files is loaded into `/usr/z1090/man`.
- ▶ A `/usr/z1090/uim` directory is also created.
- ▶ You must install the Gen2 license client later if you use Gen2 license functions. This is described in “Gen2 client configuration” on page 153

If you previously installed zPDT with the Gen2 license manager client, a new release of this license manager client is automatically installed as part of your current zPDT installation process.

## Installer options

If a zPDT *patch* is currently installed, it *must* be removed before a new zPDT release can be installed. (zPDT patches are fairly rare, but are produced when needed for specific problems.) A zPDT patch is an rpm or deb. Patches can be located with `rpm -qa | grep z109` commands and removed with `rpm -e` commands, or the Ubuntu equivalents.

The installer program has three optional functions. Using the file name in the preceding example, the functions are listed here:<sup>11</sup>

```
# ./z1090-1-10.55.05.x86_64 --refresh (reinstall current zPDT level)
# ./z1090-1-10.55.05.x86_64 --refreshall (reinstall zPDT and prerequisites)
```

<sup>9</sup> This file differs for ISV (1090) and zD&T (1091) systems.

<sup>10</sup> The “.” characters before the file name tell Linux to execute this file from the current directory.

<sup>11</sup> The same `--refresh`, `--refreshall`, and `--removeall` operands may also be used with z1091 packages.

```
# ./z1090-1-10.55.05.x86_64 --removeall    (remove zPDT and prerequisites)
```

The prerequisites mentioned here are two modules that are needed to access the USB token.

### 5.3.1 Alter Linux files

You must alter two Linux files before you can use zPDT. The first alteration is to `/etc/sysctl.conf` and involves changing a number of Linux kernel parameters. The second alteration is to the `.bashrc` file in your home directory; this adds the zPDT directories to your user's `PATH` and `LD_LIBRARY_PATH` variables. These are usually one-time changes. It is not necessary to make the changes again when upgrading to a new zPDT release.

You can manually edit the relevant Linux files or use zPDT commands to make the changes. The 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 zPDT files. If you use these two commands, you can skip the following material about manually editing these files.

#### Manually edit the files

If you want to manually edit `/etc/sysctl.conf`,<sup>12</sup> we indicate the use of **gedit** but you may use any suitable editor (such as **vi** or **leafpad**) to add the indicated lines.<sup>13</sup>

**Important:** The `shmmax` and `shmall` values installed by the `/usr/z1090/bin/aws_sysctl` script are suitable for many laptop environments where the emulated z System memory size is not more than, say, 24 GB. If your emulated z System size is much larger then you need larger `shmmax` and `shmall` values. Read the following notes carefully.

Some Linux distributions already have acceptable values for `shmmax`, `msgmnb`, `msgmax`, and `core_uses_pid`, but other distributions may 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 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 shown establishes the maximum shared memory segment size that a user can request. All z System memory, plus other zPDT work areas, is in Linux shared memory. The `shmmax` value should be at least 20% larger than the System z memory defined for the largest zPDT instance you use. The example shown (36000000000 bytes) is

<sup>12</sup> Some versions of the IBM Open Client reset these values when maintenance is applied. If this happens, you should again enter the values shown here and run `/sbin/sysctl`.

<sup>13</sup> We suggest that you do not attempt to use `vi` unless you have a basic familiarity with it.

suitable for a 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.

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 quite large and acceptable.<sup>14</sup> However, if you have multiple zPDT instances, all with large z System memory, you might exceed the default *shmall* value. If this happens you need to include a parameter such as this:

```
kernel.shmall=24000000
```

This results in the total amount of shared memory, for all users, to be 24,000,000\*4096 or about 96 GB. This value should be greater than the number of zPDT instances times the z System memory size for each instance plus about 10-20%. A number much larger than needed appears to do no harm.

**Remember:** The *shmmax* number is for the number of bytes, and the *shmall* number is for the number of pages.

The *kernel.msgmni* number, specified as 512 in this example, might need to be larger if you have many emulated I/O devices; perhaps more than 150 devices. The *msgmax* and *msgmnb* changes are not needed for some Linux releases because these are the default settings. However, including these parameters in *sysctl.conf* does no harm.

Included in the example is the *kernel.sem* parameter that controls the maximum semaphore configuration for Linux. If you have a large number of devmap devices in multiple 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.

See 13.22, “Many zPDT devices” on page 267 for more discussion about usage a large numbers of emulated I/O devices.

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.)

<sup>14</sup> 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 may need to be adjusted.

## 5.3.2 Other Linux notes

Always use the same Linux userid for zPDT operation. This Linux userid must be no longer than eight characters unless the *system\_name* statement is used in the devmap.<sup>15</sup> Multiple zPDT instances require a different Linux userid for each instance.

One of the last zPDT installation steps changes the PATH and LD\_LIBRARY\_PATH environmental variables used by Linux shells. We suggest that you do not add other directories before /usr/z1090/bin in these variables.<sup>16</sup> There are many Linux commands provided with zPDT, and these correspond to Linux file names that are accessed through the PATH variables. For example, the command **d** is used to display z System memory and registers. If you place another directory containing a file named **d** earlier in the PATH, the zPDT **d** function will not be available in the normal manner. Various internal zPDT functions assume they can access zPDT modules through PATH and LD\_LIBRARY\_PATH, so you must ensure that this is possible.

### Reboot Linux

Reboot Linux to pick up all the changes you made. Then use the **z1090instcheck** command to partly verify your environment for running zPDT. Your new PATH is needed to find the command:

```
(log in as ibmsys1)
$ 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 zPDT code correctly. Note that this command does not check any devmaps that you may have defined or copied.

## 5.4 x3270

x3270 is an open source Linux program that emulates IBM 3270 terminals<sup>17</sup>. It is not distributed with zPDT; you must obtain it yourself. You might find many different levels of x3270 on various web sites and some of these might not match the libraries available on your base Linux system. The three Linux distributions (RHEL, SLES, Ubuntu) used by zPDT normally have a version of x3270 available in their repository; for example:

```
sudo apt-get install x3270
sudo apt-get install xfonts-x3270-misc

yum install x3270
yum install x3270-x11
```

Obtaining x3270 this way should provide a version that matches your base Linux libraries. Unfortunately, the x3270 versions provided this way are often down level and might not have recent improvements. However, they will provide the basic 3270 functions.

Many 3270 emulators are available for Windows systems and these can be used with LAN connections to your zPDT system.

<sup>15</sup> The Linux userid is used as the LPAR name under zPDT, and LPAR names are limited to eight characters or less. (Only a subset of LPAR-like functions are provided by zPDT.)

<sup>16</sup> Directions for updating the PATH variables are given later in this chapter.

<sup>17</sup> There are variations of x3270 available, such as x3270c, that might be used but are not discussed here.

## 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.<sup>18</sup> With traditional 3270 keyboards this same key provides a *new line* function and the 3270 Enter key is located where the right-side Ctrl key is located on most PC keyboards.

One way to change the key mapping is to create a file (in your home directory) named `.x3270pro` (note the period as the first character of the file name):

```
! 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 keymap 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 was arbitrary. Both the Pause and the End keys are mapped to the 3270 Clear function; we did this because some keyboards no longer have a Pause key.

It is useful to remember 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.<sup>19</sup> 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.

We note that many x3270 version produce error or warning messages about font mismatches when started; these messages can be ignored if the 3270 screen function is acceptable.

## 5.5 Starting your new zPDT system

For initial testing purposes you might create the following file (which we arbitrarily name `devmap0`) in your home directory:

```
[system]
memory 8GB
3270port 3270
```

<sup>18</sup> We notice that users with strong z/VM backgrounds prefer this default key arrangement, while users with z/OS backgrounds (especially ISPF usage) prefer to reassign the keys.

<sup>19</sup> There appears to be many levels or builds of x3270 on the web and differences include the way fonts are installed or used. The author has sometimes experiment with different version to find one that includes the desired fonts.

```
processors 1

[manager]
name aws3274 1000
device 0700 3279 3274
```

You can then start zPDT with the following command:

```
$ awsstart devmap0
```

You should see startup messages from zPDT. If you have not yet installed a z System operating system, there is not much else you can do at this point. You can stop zPDT with the following command:

```
$ awsstop
```

If these steps complete without errors, your zPDT system is installed. If your token (or a connection to a remote license server) does not contain a valid license, you will not see a message saying *zPDTA license obtained*, but the startup and shutdown should work correctly.

## 5.6 Installing a different zPDT release

New zPDT releases are typically available through your Business Partner or (for IBM employees) through Resource Link. The installation procedure is the same regardless of the source. Installation is exactly the same as described earlier.

A summary of the steps is as follows:

1. Obtain the new distribution file.
2. Working as *root*, execute the distributed file. It will delete the previous release and install the new release. The process takes only a few seconds and does not disturb any of your customization.

You might want to install an older zPDT release for some reason. This is done in the same manner.





## AD-CD installation

The z System Personal Development Tool *program* provides z System functionality and associated utility programs. It does not include any z System software. z System software, including operating systems, utilities, middleware, applications, and so forth, must be obtained separately. In practice, the IBM *offerings* for ISVs (zPDT) or commercial customers (zD&T) might include the z/OS AD-CD package although it is not part of the zPDT *program*.

For software licensing purposes, a zPDT system is a z System machine and all software licensing requirements that apply to a larger z System installation also apply to a zPDT installation. This statement applies to all z System software from IBM and, we assume, applies to all z System software available from other vendors.

The discussions in this chapter assume that proper licenses have been obtained for the z System software. Licensing arrangements (and associated costs) can be complex topics and are not further addressed in this document.

**Important:** The discussions in the remainder of this document assume 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 zPDT usage and the current AD-CD releases. This is not intended as an introduction to z/OS administration.

Furthermore, we assume basic familiarity with the AD-CD z/OS package. You can find update information about the AD-CD packages at the following website:

<http://dtsc.dfw.ibm.com>

## 6.1 General principles

All current IBM z System Operating Systems (assuming proper licenses exist) are supported for zPDT usage, subject to architecture constraints described in “System z characteristics” on page 8. This includes current versions of z/OS, z/VM, and z/VSE. Linux distributions intended for z System use may be used, but all possible functions and configurations have not been extensively tested. Older version might work correctly if they are configured to work with the current z System architecture, which is z15 in the case of zPDT GA10.

## 6.2 z System operating systems

There are limitations for *installing* IBM operating systems. These limitations are related to the use of the software media and packaging techniques involved and are not limitations on the *use* of the operating systems after they are installed.

A common limitation is for any software that is distributed on tape. To install this software, your zPDT system must have a tape drive, and these are not commonly available for PC machines. Another limitation is related to any z System software that is packaged in such a way that installation requires specific z System HMC functions.

### 6.2.1 Media

In most cases (when a tape drive is not available) installation media is limited to CD, DVD, and LAN connections. (We can consider FTP or another form of downloading as *media* in this context.) Distributed files must be in formats that can be processed for zPDT. There are two meaningful formats:

- ▶ A Linux image of an emulated 3390 drive<sup>1</sup> that can be restored in the 3390 format used by zPDT. The image might be compressed (using **gzip**, for example) and would need to be uncompressed before use with zPDT. Likewise, the image might be in .tar files and would need to be extracted (and possibly uncompressed) before being used with zPDT.

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 zPDT. Whatever preliminary unpacking or uncompression is needed must be done by Linux utilities before the 3390 image can be used by a z System operating system.

- ▶ A tape image in awstape format. Such images appear as “real” tape volumes to z System Operating Systems operating on zPDT, and 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 usable by z System programs.

Another media option is to FTP a product (or other data) directly to z/OS. Some z System software is distributed in this format. zPDT must have a running z/OS and LAN connection to use this method. (Most of our discussion is for z/OS, but z/VM, z/VSE, or Linux for IBM z Systems® might be used in the same way. The point is that a working z System operating system must be installed before additional software can be sent directly to it through FTP.)

Differentiating the handling of these methods is important:

- ▶ CDs and DVDs must be processed by Linux programs (unless they contain awstape files).

---

<sup>1</sup> 3380 images could also be used, but we ignore these here.

- ▶ awstape files must be processed by z/OS (or another z System operating system), although the *transport* of awstape files can be managed by Linux through CD/DVDs, USB memory keys, or FTP.
- ▶ Direct FTPs to z/OS might be in other formats, for example in formats suitable for processing by SMP/E or the TSO XMIT command. In any event, these are z System formats and not Linux formats.
- ▶ An emulated 3390 volume (after decompression, if necessary) is a large Linux file that is meaningful only to z System software.

## 6.3 Installing a z/OS AD-CD system

The following examples use volsers (volume serial numbers) corresponding to the z/OS 2.4 (November 2019) release of the AD-CD system. These volsers tend to change in a standard pattern for new releases.

The IPL volume(s) of z/OS AD-CD releases are encrypted. The two “IPLable” volumes are usually xxRES1 and SARES1, where “xx” changes with each z/OS AD-CD update. These must be installed using the **Z1090\_AD-CD\_install** or the **Z1091\_AD-CD\_install** command. These commands decrypt the volumes and assist in implementing a signature technique that identifies the customer installing the volume.<sup>2</sup>

### 6.3.1 Specific installation instructions

There are multiple download files for current z/OS AD-CD releases, one file for each emulated 3390 volume. Documentation with each AD-CD release contains specific information about the volumes needed for that release.

Volumes other than IPL volumes are all in simple **gzip** format. AD-CD system installation might be as follows, assuming our target directory for emulated 3390 volumes is /z:

```
(We assume you are working as userid ibmsys1)
$ cd /tmp                               (or wherever you stored the downloads)
$ Z1090_AD-CD_install a4res1.zPDT /z/A4RES1      (decrypt IPL volume)
$ gunzip -c a4res2.gz > /z/A4RES2                (unzip other volumes)
$ gunzip -c a4uss1.gz > /z/A4USS1
$ gunzip -c a4sys1.gz > /z/A4SYS1
And so forth for all the volumes to be installed.
Use the Z1091_AD-CD_install command instead of Z1090_AD-CD_install if
appropriate.
```

Notice that 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; it is automatically expanded as part of the **Z1090\_AD-CD\_install** processing.

The files containing emulated volumes (and the directory containing these files) must have read and write permissions for the userid running zPDT. Assuming use of the *ibmsys1* userid, we suggest that all such files and their directories (/z, in our examples) be owned by *ibmsys1*.

<sup>2</sup> z/OS AD-CD releases prior to z/OS 2.1 were not encrypted. z/OS AD-CD releases starting with z/OS 2.1 have encrypted IPL volumes. To decrypt these you must have zPDT release 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 zPDT GA5) you should refer to earlier editions of this document.

## 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.<sup>3</sup> 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.

In all 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 simply 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.

We strongly suggest that you 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 debugging time.

## 6.3.2 IODF device numbers

We must know the device numbers (commonly known as *addresses*) used by the z/OS system. (These may be changed after the z/OS system is installed. Changing involves creating a new IODF data set, new IPLPARM member or members, and IPL of z/OS again.) Most users of the AD-CD system accept the device numbers in the IODF supplied with the AD-CD system. These are listed here:

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 defined for 120-127)
300-318	3390	3390 disks
400-40F	OSA	OSA
550-55F	3420	Round tape drives
560-56F	3480	Without COMPACT feature
580-58F	3490	Tape drives
590-59F	3590	Tape drives
600-60F	3990	Disks
700	3270	Terminal. AD-CD systems use as NIP & z/OS master console
701-73F	3277	Terminal. Normally for VTAM (TSO, CICS, etc)
900-910	3277	Terminal. Normally for VTAM (TSO, CICS, etc)
908	3270	Could be used as a z/OS console
909-91F	3277	Terminal. Normally for VTAM (TSO, CICS, etc)
A80-AFF	3390	Disks
E20-E23	CTC	3172s or CTC devices
E40-E43	CTC	3172s or CTC devices
1A00-1AFF	3390	Disks
2A00-2AFF	3390	Disks
3A00-3AFF	3390	Disks

Most of the addresses are three hex digits, due to historical reasons. Both the AD-CD z/OS system and zPDT system can work with four-digit addresses. These addresses have been stable for many releases of the z/OS AD-CD system; however, it is possible they might change in future releases.

<sup>3</sup> The volser is written by the ICKDSF utility program (for a new volume), or is already present in a volume that was downloaded or taken from a DVD.

In principle the 3390 IPL volume, for example, could be mounted at any address defined as a 3390. By AD-CD convention, the IPL volume and the volume containing the IODF and other key data sets are usually mounted at addresses A80 and A82:

<b>VOLSER</b>	<b>ADDRESS</b>	<b>Purpose</b>
A4RES1	A80	IPL volume and key z/OS libraries
A4SYS1	A82	Spool space, LOGGER data sets, VSAM, etc

The A80 and A82 addresses are used in AD-CD 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 simply start at address A80 and increment it sequentially for each additional volume.

### 6.3.3 zPDT control files

Before the AD-CD system can be used an appropriate devmap must be created. A basic example is shown here:

```
$ cd /home/ibmsys1
$ gedit aprof23                                     (this is an arbitrary file name)
[system]
memory 9000m                                       # define 9000 MB z System
processors 1                                       # use 2 or 3, if appropriate
3270port 3270                                     # 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/A4RES1                  # (The "A4" prefix is for the
device 0a81 3390 3990 /z/A4RES2                  # z/OS AD-CD released in November
device 0a82 3390 3990 /z/A4SYS1                  # 2019. Later releases will have a
device 0a83 3390 3990 /z/A4CFG1                  # different prefix)
device 0a84 3390 3990 /z/A4USS1
device 0a85 3390 3990 /z/A4USS2
device 0a86 3390 3990 /z/A4PRD1
device 0a87 3390 3990 /z/A4PRD2
device 0a88 3390 3990 /z/A4PRD3
device 0a89 3390 3990 /z/A4PAGA
device 0a8a 3390 3990 /z/A4PAGB
device 0a8b 3390 3990 /z/A4PAGC
device 0a8c 3390 3990 /z/A4USR1
#(continue with whatever additional volumes 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 you will use when starting zPDT so that you do not need to enter a full path name when using it.

We suggest you do not define OSA devices for your initial z/OS startup. The OSA definitions can be a little more complex and we suggest you verify that your basic z/OS system is operational first.

### 6.3.4 IPL and operation

Start zPDT with an **awsstart** command. Among other functions this starts the 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 already exist) and several zPDT-related directories below it.

```
$ 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 incident(s), 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:<sup>4</sup>

```
$ x3270 -port 3270 localhost &
$ x3270 -port 3270 localhost &
$ x3270 localhost:3270 & (another way to specify a port number)
```

Consider the following information:

- ▶ x3270 is the name of the program.
- ▶ 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; this is indicated by the localhost operand.
- ▶ The ampersand (&) causes the x3270 program to execute in the background, leaving the Linux window free for additional commands. We can recall and execute the **x3270** command repeatedly to create multiple 3270 sessions.

The 3270 window displays identification lines if there has been no data sent to it by the z 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 and these are discussed in 3.3.3, “The aws3274 device manager” on page 42. The File and Options menus at the top of the x3270 window can be used for a variety of functions. Changing the font size (using the Options menu) has the effect of changing the 3270 window size.

The 3270 session for the z/OS console (address 700 for the AD-CD system) should be ready before z/OS IPL. Next, issue the appropriate IPL command in the Linux window:<sup>5</sup>

```
$ ip1 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 AD-CD system (or an IPL after a long period of non-use, or a changed z System serial number) you might see messages similar to this one:

<sup>4</sup> These instructions assume you have x3270 installed and are using it on the same PC that is running zPDT.

<sup>5</sup> This example assumes you have mounted the IPL volume at address A80 and the volume containing the IODF and IPL parameters at address A82. By AD-CD convention, the “cs” IPL parameter causes a JES2 cold start.

IXC420D REPLY I TO INITIALIZE SYSPLEX ADCDPL, OR R TO REINITIALIZE XCF

If this message is issued, go to the 3270 session displaying the message and enter the following command:

**r 00,i**

After VTAM is started, the VTAM logo should display on the other 3270 sessions.<sup>6</sup>

There is usually some documentation for each AD-CD release that provides details about different IPL parameters and TSO logon procedures. A brief summary for a recent z/OS AD-CD system is shown here:

IPLparm	LogonProcedure	Purpose
0A82CS	ISPFPROC	Basic IPL without DB2, etc. Cold start JES2.
0A8200	ISPFPROC	Subsequent basic IPLs. Warm start JES2
0A82DC	DBSPROC	Initial IPL for Db2 V12, etc

User IBMUSER is always present on z/OS and is typically used for initial TSO logons. The initial password for IBMUSER should be published with the AD-CD 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 AD-CD releases. The procedures are in the ADCD PROCLIB data set.

The Linux command window that was used for the **awsstart** command should be kept open, if possible. Asynchronous messages from zPDT are sent to this window. You can enter zPDT commands from other windows, but it is possible that you might miss significant messages that are sent to the original window.

### 6.3.5 Shutting down

z/OS should be shut down cleanly, if possible. Enter the **s shutdown**<sup>7</sup> command at the z/OS console and reply to any messages produced. The message ALL FUNCTIONS COMPLETE indicates that JES2 can now be stopped with the command **\$PJES2**. After JES2 ends, System z operation can be stopped. The zPDT system is stopped with this command in the Linux window:

**\$ awsstop**

This produces several messages. It might be necessary to press Enter to obtain the Linux prompt. Any 3270 windows may be closed at this point.

### 6.3.6 Startup messages

Messages such as the following are produced by the **awsstart** command:

```
AWSSTA014I Map file name specified: aprofa2
0 Snapdump incident(s), 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
```

<sup>6</sup> If the 3270 session displays a message Unsupported Function, simply use the 3270 Clear key to obtain the initial VTAM display. Some TN3270e emulators encounter this initial message and others do not.

<sup>7</sup> This **shutdown** command is not a standard z/OS function; additional command names might be present. These trigger VTAMAPPL scripts to issue various commands involved in stopping z/OS.

Glance at these messages, because SNAP dump incidents are indications of an internal zPDT error, and if you want to work with your zPDT support, you will need this data. The number of bytes used for various logs and dumps is usually not significant unless it becomes too large. If the numbers displayed become too large (many megabytes) and if you are not actively working on a problem with your zPDT support organization, you might want to clean up these files. To do this, add the `--clean` option the next time you issue an `awsstart` command:

```
$ awsstart aprof11 --clean
```

You can get the `--clean` behavior every time by setting a Linux shell environment variable `Z1090_CLEAN=YES`; however, we do not suggest doing this because it might easily result in the removal of important debugging information in the event of a zPDT failure.

### 6.3.7 Local volumes

For our examples, we created a Linux disk partition mounted at `/z`. The process for adding your own 3390 volumes is outlined here:

1. Create the emulated 3390 volume using a zPDT utility:

```
$ a1cckd /z/WORK01 -d3390-3           (assuming 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 zPDT with the updated devmap.
4. IPL z/OS with the new volume present. z/OS will detect an uninitialized volume and vary it offline.
5. Create and run an ICKDSF job to initialize the volume:

```
//BILLX JOB 1,OGDEN,MSGCLASS=X
// EXEC PGM=ICKDSF,REGION=1M
//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

We can install multiple z System Operating Systems, limited only by the disk space we have available. Every emulated 3390-3 volume uses approximately 2.8 GB of disk space and 3390-9 volumes use about 8.6 GB.

It is important to distinguish between *installing* additional emulated 3390 volumes (perhaps with a variety of operating systems), and *using* the volumes. We can, of course, only IPL a single system at any one time in a zPDT instance.<sup>8</sup> The volumes that might be “seen” by that system depend on several factors:

---

<sup>8</sup> This statement ignores the possibility of running multiple z/OS guests under z/VM.

- Does the current devmap contain all the desired volumes?

You can have multiple devmaps, each with a different selection of emulated volumes and assigned addresses, but we can have only one devmap specified when we start a zPDT instance. You cannot change the active devmap while zPDT is running.<sup>9</sup> (You can change the devmap file after zPDT is started, but this file change has no effect on the running 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 AD-CD IODF would not “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 may not be present.

You may 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 typically associated with an operating system.

For example, all the AD-CD documentation uses A80 as the IPL address for a z/OS AD-CD system. We can have two (or more) AD-CD systems represented in the devmap at the same time, but only one volume can have address A80 during any single execution of zPDT. This does not prevent us from IPLing any of the (multiple) AD-CD systems present, but we need to specify the correct address. An example might make this clearer:

Address	VOLSER	Purpose
A80	A4RES1	IPL volume for z/OS 2.4 AD-CD system
A81	A4RES2	Libraries for z/OS 2.4 AD-CD system
A82	A4SYS1	Paging, spooling, VSAM for 2.4 AD-CD system
...		
A90	D3RES1	IPL volume for z/OS 2.3 AD-CD system
A91	D3RES2	Libraries for z/OS 2.3 AD-CD system
A92	D3SYS1	Paging, spooling, VSAM for 2.3 AD-CD system
...		

Assuming our devmap is configured for these addresses, we can **ip1 A80 parm 0A82CS** to run the 2.4 system or we can **ip1 A90 parm 0A92CS** to run the 2.3 system.<sup>10</sup> In either case, the running z/OS system can access all the volumes of both z/OS systems. This is convenient for migration purposes. The volumes are readdressed by simply changing the addresses in the devmap.

We can run multiple z System Operating Systems at the same time by starting multiple zPDT instances, but this requires more resources (especially PC memory).

<sup>9</sup> This is not completely true. We can change the volume mounted on an emulated disk drive or tape drive with the **awsmount** command.

<sup>10</sup> Yes, we know these levels of z/OS are ancient and do not work on current zPDT system, but they illustrate the point being made.





# LANs

LAN setup can become complicated, partly because we must deal with physical LAN hardware and interfaces in the underlying Linux as well as z System interfaces. At the z System level, zPDT provides emulated OSA functions. OSA functions are closely tied to CHPID usage (on a large z System) and we must deal with CHPID-like details at the zPDT level. A zPDT system can use more than one LAN adapter, although this is unusual. The following discussions are based on z/OS usage, but similar considerations are present with other z operating systems.

In this discussion we consider only ethernet connections. We use “LAN” and “ethernet” interchangeably, and consider this to include wired or wireless ethernet and ethernet “tunnel” operation, where appropriate.

**Attention:** zPDT provides OSA operation in both QDIO and non-QDIO modes. We recommend QDIO mode unless you have a specific need for non-QDIO operation. Appendix B., “Non-QDIO OSA” on page 357 provides information about non-QDIO operation and non-QDIO operation is not further considered in the current chapter.

For our purposes a wireless connection is treated simply as another ethernet adapter and can be used as such by Linux TCPIP and OSA emulation. Generally, wireless connections are slower than wired connections and this might be relevant for large FTP transfers. Also, wireless connections are more prone to disruptions than wired connections. This can cause problems if, for example, the link to the MVS console is disrupted.

Our examples include basic use of Linux `ip` commands (older Linux) and `firewall-cmd` commands (some recent Linux distributions). You might need more detailed use of these commands and we do not attempt to describe this.

## 7.1 Overview of LAN usage

Several key factors permeate this discussion:

- ▶ A given PC ethernet adapter can be used 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 (via zPDT OSA) is using the same Ethernet adapter. This is a common configuration for zPDT.

- ▶ We have the base Linux TCP/IP and z/OS TCP/IP.<sup>1</sup> These two functions operate separately. *Always* be aware of *which* TCP/IP is under discussion.
- ▶ You do not need any z/OS TCP/IP operation for one mode of 3270 access to z/OS. Access can be through the aws3274 device manager and appears as local, channel-attached terminals to z/OS. OSA definitions are not used. This is our first scenario.
- ▶ z/OS and the base Linux cannot communicate with each other through the same hardware LAN adapter. Both can share the same hardware LAN adapter for all TCP/IP functions *except* communicating with each other. zPDT implements a *tunnel*<sup>2</sup> pseudo-LAN to bypass this restriction.
- ▶ Standard z/OS is not a DHCP client. You cannot simply plug z/OS into 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 **x3270** program is frequently mentioned in this discussion. This is an open-source program that is not distributed by IBM. Various versions may be found on many web sites. To the best of the author's knowledge, it is the only common, freely-available Linux-based 3270 emulator.

The discussions here involve both the Linux TCPIP functions (using the internal Linux ethernet support) and z/OS TCPIP (using OSA interfaces). These are two separate TCPIPs and *you must always be aware of which one is being discussed*. Without this awareness the discussions can be very confusing.

### 7.1.1 zPDT 3270 interfaces

There are two 3270 interfaces for z/OS<sup>3</sup>:

- ▶ The aws3274 device manager accepts TN3270e connections<sup>4</sup> from the *Linux* TCP/IP network.<sup>5</sup> The Linux TCP/IP port number for this connection is specified in the 3270port parameter in the devmap. z/OS sees these 3270 sessions as local, channel-attached, non-SNA, DFT terminals. Such terminals are suitable for z/OS operator consoles and VTAM use. z/OS TCP/IP is not involved and does not need to be running.
- ▶ z/OS TCP/IP provides TN3270e connections via OSA connections. Terminals connected this way are not usable as z/OS operator consoles. TN3270e connections through z/OS TCP/IP are routed through VTAM and may be used as TSO terminals, IBM CICS terminals, and so forth. The OSA-Express functions are emulated by the awsosa device manager.<sup>6</sup>

<sup>1</sup> We usually discuss z/OS throughout this material, but equivalent statements can apply to other z System operating systems.

<sup>2</sup> 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. It provides a pseudo LAN adapter.

<sup>3</sup> As a reminder, we ignore non-QDIO operation in this chapter. Non-QDIO usage with SNA protocols potentially could be another 3270 interface.

<sup>4</sup> A TN3270 connection (as opposed to a TN3270e connection) will be accepted, but extended data stream capabilities are not present and some z/OS functions might not work correctly.

<sup>5</sup> The 3270 sessions might be on the base Linux PC and/or on other PCs connected to the base Linux PC via the network.

<sup>6</sup> We describe this as an OSA-Express5s device manager, but this description is only approximate. This device manager has attributes of the original OSA, OSA-Express, and OSA-Express5s channels available on larger System z machines.

## Specific limitations

z/PDT, used with the current z/OS system, has specific limitations:

- ▶ GVRP (Generic VLAN Registration Protocol) 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 zPDT.

## 7.2 OSA paths

The Ethernet adapters used by awsosa are identified by the *path* parameter<sup>7</sup> in the devmap or possibly by an interface name. The zPDT **find\_io** command displays the current Linux Ethernet device configuration. A default zPDT path is assigned for most interfaces. Figure 7-1 shows an example of using a **find\_io** command.

\$ <b>find_io</b>								
FIND_IO for "ibmsys1@linux-8jfl"								
Path	Interface Name	Current State	MAC Address	IPv4 Address	IPv6 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	*	*			
...								
Path	Interface 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 may lead to poor zPDT Performance, please refer to your 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 **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 device map must have a unique path specified.
- ▶ The path names displayed by **find\_io** are the default values and 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 device map.

<sup>7</sup> In this context we could use the term *CHPID* instead of *path* to more closely relate to large z System terminology.

- ▶ The Ethernet interface names are listed in alphabetical order, with exceptions. Nomenclature such as `ethx`, `em[1,2,3,4]`, `emnsnn`, and `p<slot>p<port>` cause the motherboard interfaces to be listed first.
- ▶ Wireless interfaces with names something 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.
- ▶ Interface names `br[x]` and `virbr[x]` might be listed but not assigned default paths because these interfaces are not generally used for OSA Express emulation.<sup>8</sup>
- ▶ Interfaces for `pan[x]` might be listed and assigned default path names but have not been investigated or tested for zPDT usage.
- ▶ Linux alias addresses are not listed.

The other **find\_io** details (status, MAC address, IP addresses) are informational. The IP addresses are those used by the base Linux machine. z/OS (or another z System operating system) running within a zPDT instance would use different IP addresses for these interfaces.

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 MTU (maximum transmission unit) listed is for Linux, not the hosted z operating system.

The devmap **name** statements for OSA devices might look like one of these:

```
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 two-byte hexadecimal path name (FF). (This arbitrary path name must not conflict with other path names in the device map.)
- ▶ The forth example illustrates that any path can be assigned to an interface.
- ▶ Only one path may be assigned to an interface.
- ▶ The MAC addresses for tap devices are arbitrary and not usually meaningful.

The IP address used by z/OS for OSA Express operation is set by the TCPIP PROFILE parameter for z/OS TCPIP, or the equivalent when using a different z System operating system. These addresses are not shown by **find\_io** because they are not known to Linux.

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 as follows:

CHPID	LinuxName	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. <sup>9</sup>			

<sup>8</sup> We have seen cases where bridge devices seem to create routing problems.

In general, the multiple tunnel interfaces are intended for use with multiple zPDT instances. The same tunnel interface cannot be shared by multiple 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 could 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

LAN setup can become complex and many variations are possible. We have selected five basic scenarios as possible starting points. We *strongly* suggest that your initial zPDT usage be with scenario 1, which has no z System TCP/IP functions.<sup>10</sup> The second scenario is then a simple migration from the first one. The third, fourth, and fifth scenarios provide different ways to connect zPDT functions to an external network. The key difference between these last three scenarios is whether you have a fixed IP address that can be used with your z/OS (or z/VM, or z/VSE, or Linux for z Systems).

This chapter does not address more complex LAN use, such as might be used for multiple guests under z/VM. Again, we strongly suggest you start with the basic scenarios described in this chapter. After working through these you should then be familiar with the elements of LAN use that are unique to zPDT.

**Important:** LAN setup is not part of the 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”. This has implications about external routing, which physical LAN segment you use, domain and host names, and other factors.

## 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).
- ▶ z/OS devices involved must be defined as OSA devices in the z/OS IODF.

Recent z/OS AD-CD 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 “More complete QDIO example” on page 138.

- ▶ A TRLE definition is needed in VTAMLST, 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 be started. 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 TCPIP PROFILE parameters in z/OS is an arbitrary name. In the TRLE this is the PART name; in the z/OS TCPIP 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.

<sup>9</sup> The path names are expressed in hex and the Linux interface names have decimal suffixes.

<sup>10</sup> If you follow the basic installation instructions in this document, you will be close to this environment.

The example in 7.11, “More complete QDIO example” on page 138 uses PORTA and PORTB as these arbitrary names.

- ▶ The z/OS TCPIP PROFILE uses a 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/VSE or z/VM or Linux for System z could be used with appropriate adjustments. QDIO usage in z/OS requires parameters in VTAMLST, and these are included in the setup examples.

The five scenarios are 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 could be connected to a larger LAN; this is transparent to z/OS. Emulated 3270 sessions from the base Linux or from the LAN 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 via a tunnel function.<sup>11</sup> All z/OS TCP/IP activity is directed to the tunnel. This allows TCP/IP connections between the base Linux and z/OS, and these might be used for FTP, Telnet (to UNIX System Services), TN3270e connections directly to z/OS TCP/IP, and so forth. The base Linux could be connected to an external LAN, but this is transparent to z/OS and external LAN connections cannot be made to z/OS.
3. The same basic setup as scenario 2, but with additional customization to enable a simple NAT function in the base Linux. This permits TCP/IP connections from z/OS to the external LAN, but not from the external LAN to z/OS. (That is, only outgoing TCP/IP sessions may be initiated. With additional NAT/iptables customization, incoming TCP/IP connections from the external LAN to z/OS could be handled. This additional customization might involve non-standard port numbers for either Linux or z/OS.)
4. Instead of the NAT functions 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 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 zPDT require the use of 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 zPDT `find_io` command. This command might not display information for tap devices until after zPDT is started at least once with a tunnel definition included in the devmap.

### 7.5.1 Scenario 1

Scenario 1 is illustrated in Figure 7-2.

<sup>11</sup> The correct terminology is “connect via a Linux tap interface.” However, we use the term *tunnel* in a generic sense to describe this connection.

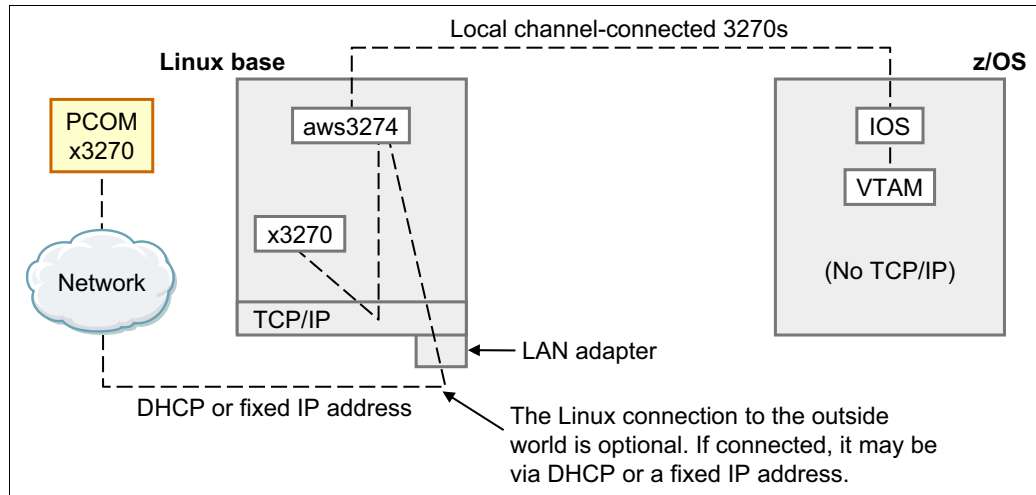


Figure 7-2 Scenario 1 connectivity

With this option, no z/OS TCP/IP setup is required and z/OS TCP/IP does not need to be active. (The AD-CD z/OS system starts TCP/IP by default. You could remove the associated **start** statement in a 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 PCOMM (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.<sup>12</sup>

No OSA definitions are needed in the devmap. The relevant devmap definitions are for the 3270 port and for several local 3270 devices.

```
[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 AD-CD systems
device 0701 3279 3274      #Other systems may want different addresses
device 0702 3270 3274
...
```

Based on this example, connections from the base Linux might start as follows:

```
$ x3270 -port 3270 localhost &
```

A connection from the external LAN might be started as follows:

```
$ x3270 -port 3270 9.111.222.123 &
```

This assumes the address 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.<sup>13</sup>

<sup>12</sup> These are often known as *file transfer* functions in the 3270 emulators.

<sup>13</sup> However, if your Linux is connected to a local router the DHCP address may be valid only in the local network created by that router. IP addresses in the 10.xxx.xxx.xxx and 192.168.xxx.xxx range are usually in this category.

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 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 you defined for aws3274 connections). If your firewall is based on iptables (as is common for most current Linux releases), commands such as the following might be used:

```
$ 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 would be 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 document.

## 7.5.2 Scenario 2

Scenario 2 builds on scenario 1, and adds a direct TCP/IP connection between z/OS and the base Linux, as shown in Figure 7-3.

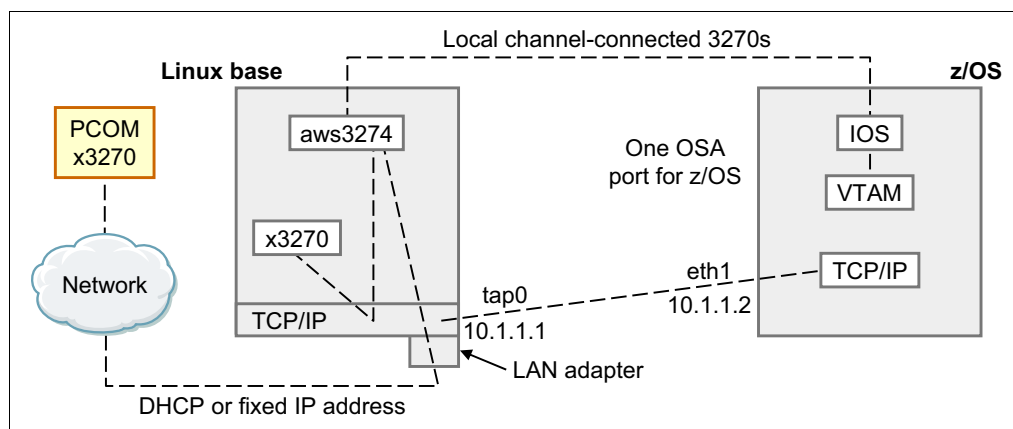


Figure 7-3 Scenario 2 connectivity

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 shown are arbitrary, but we suggest using non-routable addresses on an isolated subnet. The tap interface (and associated IP address) are created automatically when zPDT is first started (assuming 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.

AD-CD z/OS systems include OSA devices starting at device number 400. When using the QDIO interface to the emulated OSA-Express5s function, the key parameters might look like the following example:

### Devmap

(The 3270 port and aws3274 device manager definitions 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<sup>14</sup>

```

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
...
BEGINRoutes
;      Destination Subnet Mask      FirstHop      Link Size
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 connected to Linux and the “tunnel LAN” between Linux and z/OS are completely 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 may be used. Examples (from the Linux side) include:

```

$ x3270 -port 3270 localhost &           (connect via “local 3270” channel)
$ x3270 10.1.1.2 &                       (connect via z/OS TCP/IP)
$ ftp 10.1.1.2                           (connect to z/OS FTP)
$ telnet 10.1.1.2 102315                (connect to z/OS UNIX System Services)

```

From the z/OS TSO side, we might use a command such as this one to connect to FTP on the base Linux. (This assumes you have FTP 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 TCPIP PROFILE examples to match the recent z/OS AD-CD releases. You can use the INTERFACE form instead. For example:

```

INTERFACE LAN1
  DEFINE IPAQENET
  IPADDR 10.1.1.2
  PORTNAME PORTA

```

## Tunnel IP addresses

The IP addresses used for the tunnel (10.1.1.1 and 10.1.1.2 in the examples) are not related to any other IP addresses 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 you might use. These tunnel addresses are solely for use between the base Linux and TCP/IP stacks running within the zPDT environment.

<sup>14</sup> Remember that the VTAMLST ATCCONxx member must point to this TRL member.

<sup>15</sup> The AD-CD z/OS system uses port 1023 for a simple Telnet connection to UNIX System Services. You might need to define this functions as described in 12.20, “TELNET” on page 245. SSH could be used instead of Telnet.

The IP address for the base Linux side of the tunnel defaults to 10.1.1.1 (for the first tunnel OSA), but may 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 connected to the local side of this router.

### 7.5.3 Scenario 3

Scenario 3 is depicted in Figure 7-4.

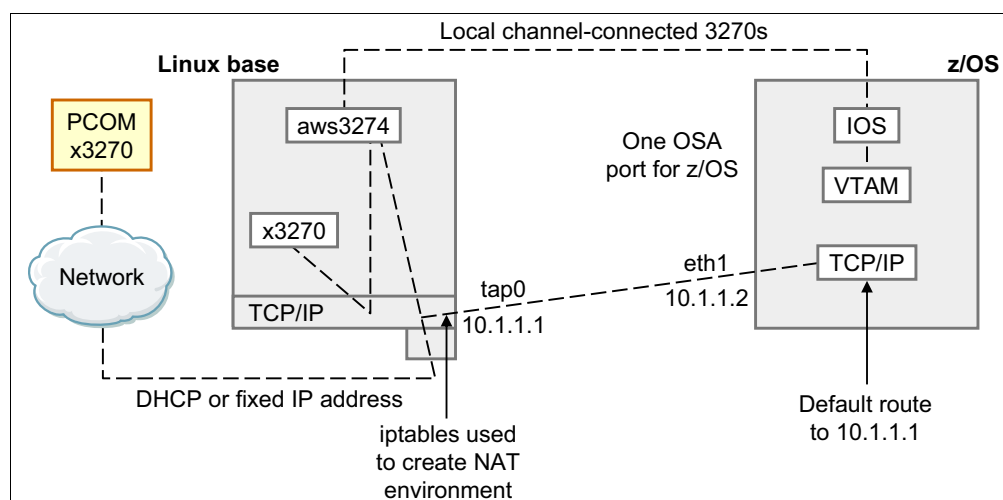


Figure 7-4 Scenario 3 connectivity

We can take the scenario 2 setup and extend it to connect z/OS to the external LAN by using a NAT<sup>16</sup> function in the base Linux. This requires a more complex setup. However, it 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 this is not an externally assigned address; it is visible only internally in our local Linux system.

The following text provides an example to do this dynamically (through commands each time the system is started), using **iptables** commands or **firewall-cmd** commands (whichever 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 zPDT home directory, as shown here. We 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'
```

<sup>16</sup> NAT is Network Address Translation.

```

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 located 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`; the author's current notebook uses the name `enp0s25`, for example. There are many ways to display the NIC name, including the `zPDT find_io` command.

**Important:** The specific **iptables** commands listed here were suitable for the Linux we used at the time 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 shown for scenario 2. Assuming 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)
# ./masq                   (execute the command script we just created)
# exit                     (exit from root)

```

### **Working with firewall-cmd**

If your base Linux uses the `firewalld` function issue the following Linux commands:

```

# firewall-cmd --zone=external --change-interface=eth0
# firewall-cmd --add-port=3270/tcp --zone=public

```

The *external* zone (of the `firewalld` 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.

### **Testing**

You should be able to access external network sites from z/OS. For example, at the time of writing the `www.ibm.com` site is IP address 104.107.41.53. From a TSO command line, try the following command:

```
ping 104.107.41.53
```

You should be able to **ping** any Internet sites that are known to respond to pings,<sup>17</sup> if you can find their numeric IP address.

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; this provides a *hole* in the standard Linux firewall for using the local 3270 connections. Linux port 3270 (using our examples) is the `aws3274` device manager and the connection results in a z/OS 3270 session for the external user.

<sup>17</sup> Our informal tests indicate that most common Internet sites no longer respond to pings. You can verify your results by issuing pings from the base Linux system.

## Incoming connections

The nat setups have complications for incoming (from the external LAN) connections. Remember that external systems see only the IP address of the base Linux. (This is probably a DHCP-assigned address, and the address must be communicated to external users in some manner.) If an external user attempts to connect to port 23, for example, does the user want Linux port 23 or z/OS port 23? (This assumes the user can get through the Linux firewall, which is another complication.) Port 23 is a well-known (default) port number for Telnet connections (including TN3270e Telnet).<sup>18</sup>

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 simply disallow port 23 connections to either Linux or z/OS. (The issue applies to all port numbers; 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:

```
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 would have two paths for a TN3270e connection (assuming the Linux IP address is 192.168.1.2):

\$ x3270 192.168.1.2 &	<i>Forwarded to z/OS TCP/IP port 23</i>
\$ x3270 192.168.1.2:3270 &	<i>Connect to Linux port 3270 (aws3274)</i>

Be aware that after this **iptables** command is issued, we 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<sup>19</sup> and can be used by both Linux and z/OS. z/OS must have an external assigned, fixed IP address for this to work. Our example uses address 192.168.1.61, but this is just an example.<sup>20</sup> You *must* have an IP address that is suitable for whatever network you are using. If you are confused by this please contact someone with appropriate network knowledge.

Figure 7-5 illustrates this configuration. The figure shows two logical connections to the external network, but this is accomplished by a single physical cable connection.

<sup>18</sup> Some Linux systems have completely dropped Telnet service (that listens on port 23); our comments apply to all ports.

<sup>19</sup> You could use a second PC network adapter, but care is needed to avoid external routing loops. If you decide to use a second adapter, we suggest you first debug your network environment using a single adapter.

<sup>20</sup> Remember that 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, such as our examples illustrate.

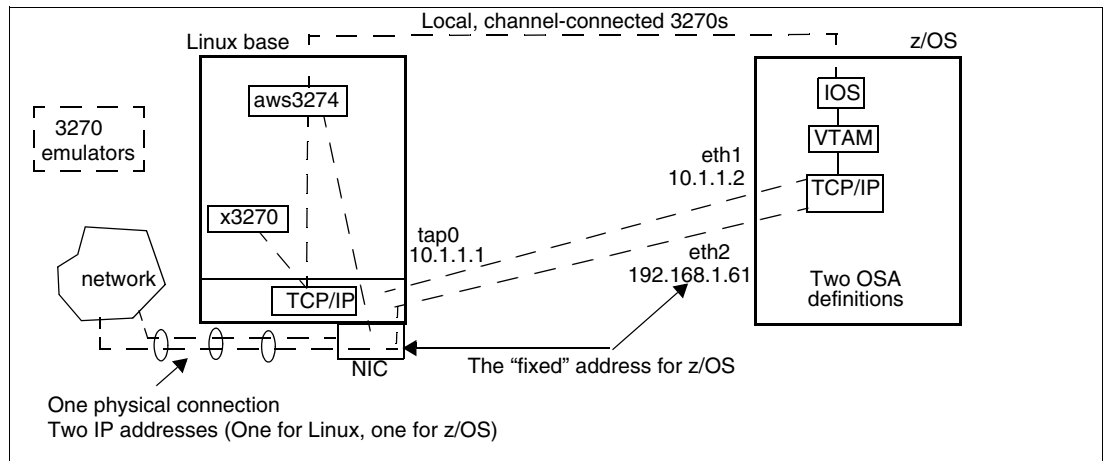


Figure 7-5 Scenario 4 connectivity

With this configuration the IP functions of z/OS and the base Linux are quite separate. There are three separate IP connections involved: (1) Linux to the external LAN, (2) the 10.x.x.x tunnel, and (3) z/OS to the external network.

The various definition files might contain the following details:

#### Devmap

(The 3270 port and aws3274 device manager definitions 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 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. That is why we show two OSA definitions and connections in this example.

## 7.5.5 Scenario 5

Figure 7-6 shows connectivity for this scenario.

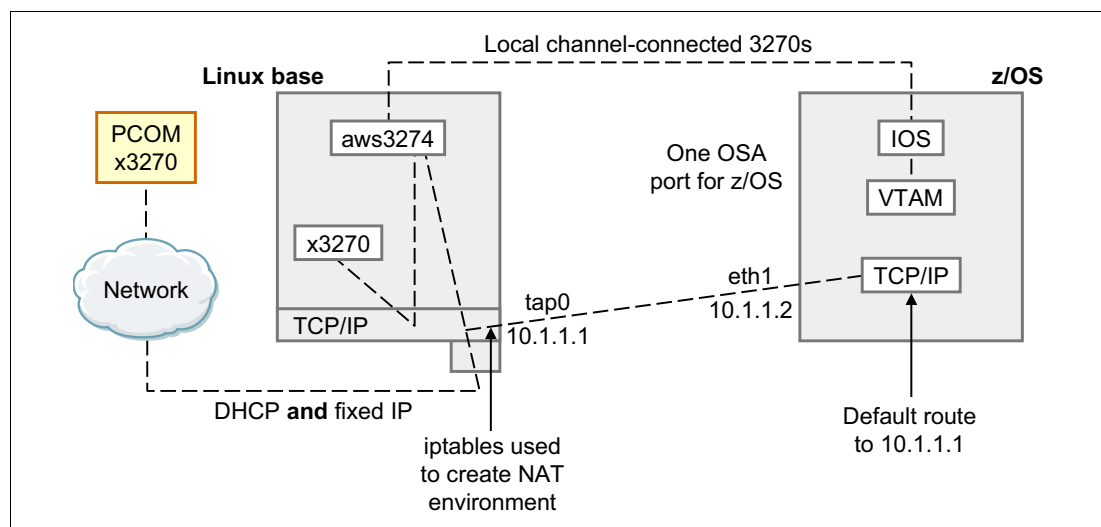


Figure 7-6 Scenario 5 connectivity

We 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, as well as outgoing connections.

The following text describes how to do this dynamically. The example is only for iptables; users of firewall-cmd must devise a similar scheme. Note that this method uses an IP alias address in the base Linux.

The first step is to create a Linux file in the zPDT home directory. We use a more elaborate script file here to better allow it to be expanded in the future. We named this file *nat2*. You must, of course, substitute your appropriate IP addresses in the script.

```

$ cd ~
$ touch nat2

```

```

$ gedit nat2          (the following lines start in column 1 in the file21)
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_AO_INTERFACE=eth0
CHPID_AO_EXTERNAL_IP=192.168.1.80          (your assigned IP address)
CHPID_AO_EXTERNAL_BC=192.168.1.255        (broadcast address for it)
CHPID_AO_EXTERNAL_NM=255.255.255.0        (net mask for it)
CHPID_AO_VIRTUAL_IP=10.1.1.2

echo 1 > /proc/sys/net/ipv4/ip_forward
echo 'IP forwarding set'
iptables -t nat -F
echo 'nat table flushed'

echo 'External IP address for System z is ' $CHPID_AO_EXTERNAL_IP
echo 'Real LAN interface is ' $CHPID_AO_INTERFACE
echo 'Tap (tunnel) address for System z is ' $CHPID_AO_VIRTUAL_IP
echo 'External netmask and broadcast address are ' $CHPID_AO_EXTERNAL_NM \
$CHPID_AO_EXTERNAL_BC

ifconfig $CHPID_AO_INTERFACE:0 $CHPID_AO_EXTERNAL_IP netmask \
$CHPID_AO_EXTERNAL_NM broadcast $CHPID_AO_EXTERNAL_BC up

iptables -t nat -A POSTROUTING -o $CHPID_AO_INTERFACE -s \
$CHPID_AO_VIRTUAL_IP/32 -j SNAT --to $CHPID_AO_EXTERNAL_IP

iptables -t nat -A PREROUTING -i $CHPID_AO_INTERFACE -d \
$CHPID_AO_EXTERNAL_IP/32 -j DNAT --to $CHPID_AO_VIRTUAL_IP

echo 'Done. Please exit from root'

```

Use the same devmap and z/OS PROFILE parameters shown for scenario 2. Assuming 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      (execute the command script we just 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<sup>22</sup> and connect to z/OS by using its fixed address.

## 7.5.6 Scenario comparison

Table 7-1 summarizes the characteristics of the five scenarios. Note carefully the *from* and *to* words in the descriptions.

<sup>21</sup> Four lines in this file end with a back slash (\) 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 back slash).

<sup>22</sup> You can also have an assigned, fixed address for your base Linux.

Table 7-1 Scenario characteristics

Characteristic	Scenario:	1	2	3	4	5
Number of local 3270 sessions <i>from</i> base Linux (including external LAN) <i>to</i> z/OS (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” needed to run in base Linux (iptables)		0	0	1	0	1
External LAN connection <i>from</i> or <i>to</i> z/OS (not counting <i>local</i> 3270 sessions) <sup>a</sup> ?		No	No	Yes <sup>b</sup>	Yes	Yes
TN3270, FTP, Telnet <i>from</i> local Linux <i>to</i> z/OS TCP/IP. FTP <i>from</i> z/OS <i>to</i> local Linux		No	Yes	Yes	Yes	Yes
TN3270e <i>from</i> external LAN <i>to</i> z/OS		No	No	Maybe <sup>c</sup>	Yes	Yes
FTP <i>from</i> z/OS <i>to</i> external LAN		No	No	Yes <sup>d</sup>	Yes	Yes
Externally assigned IP address needed for z/OS		No	No <sup>e</sup>	No <sup>e</sup>	Yes	Yes
Telnet connection <i>to</i> UNIX System Services		No	Yes, only from base Linux	Yes, only from base Linux	Yes	Yes
Browser connection <i>from</i> base Linux <i>to</i> z/OS		No	Yes	Yes	Yes	Yes
Browser connection <i>from</i> external LAN <i>to</i> z/OS		No	No	Maybe <sup>c</sup>	Yes	Yes
Browser or FTP connection <i>from</i> external LAN <i>to</i> 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 may be used unless additional iptables commands are used. The additional command functions are likely to require the use of non-standard port numbers.

c. This is an incoming connection and is not possible in the most basic iptables or firewall-cmd example. Incoming connections may be accepted when an additional iptables setup is used.

d. This is an outgoing connection and may be used. A *passive* FTP connection may 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 require an assigned, fixed address for z/OS. Scenario 3 has complications for incoming connections (from the external LAN) to z/OS; 4 and 5 do not have this restriction. The primary difference between 4 and 5 is where the external LAN interface appears; with scenario 4 it is defined to z/OS; with scenario 5, it is defined in the base Linux.

## 7.5.7 z/OS resolver

The most recent AD-CD system<sup>23</sup> (at the time of writing) starts a resolver function as follows. The “xxxx” qualifier in the dataset names changes with each z/OS AD-CD release.

<sup>23</sup> The November 2016 z/OS AD-CD.

- ▶ The job is in `ADCD.xxxx.PROCLIB(RESOLVER)`. This has the `SETUP DD` statement pointing to `ADCD.xxxx.TCPPARMS(GBLRESOL)`, which contains the primary controls for the resolver.
- ▶ The `GBLRESOL` member contains the following lines, with our modifications highlighted in bold:

```

DEFAULTTCPIPDATA('ADCD.xxxx.TCPPARMS(GBLTDATA)')
GLOBALTCPIPDATA('ADCD.xxxx.TCPPARMS(GBLTDATA)')
GLOBALIPNODES('ADCD.xxxx.TCPPARMS(GBLIPNOD)')
DEFAULTIPNODES('ADCD.xxxx.TCPPARMS(GBLIPNOD)')
COMMONSEARCH
CACHE
CACHESIZE(200M)                                <---this is probably too large
MAXTTL(2147483647)
UNRESPONSIVETHRESHOLD(15)                    <---we changed this to 15

```

- ▶ The `GBLTDATA` member (leaving out the comments) contained these lines:

```

TCPIPJOBNAME TCPIP
SOW1: HOSTNAME SOW1
DOMAINORIGIN  OGDEN.ITSO.IBM.COM             <---use your own domain name !
NSINTERADDR 167.206.10.178                   <---find your own name server
NSPORTADDR 53
RESOLVEVIA UDP
RESOLVERTIMEOUT 15
RESOLVERUDPRETRIES 1
ALWAYSUTO NO

```

- ▶ The `GBLIPNODE` member contains something like this line:

```

10.1.1.2 SOW1.OGDEN.ITSO.IBM.COM             <---use your own node name

```

Before using the resolver for “real” work on the Internet your z/OS system must have access to the “real” Internet. This could be done in multiple way, including one of the following:

- ▶ Use the **iptables** script shown in 7.5.3, “Scenario 3” on page 124 to create a nat connection to the internet via the base Linux system.
- ▶ Instead of direct **iptables** commands, use the **firewall-cmd** command to connect to a zone providing nat; for example, **firewall-cmd --set-default-zone=external**.
- ▶ Use a direct internet connection, as described in scenario 4.

Our operational procedure was similar to these steps:

1. Create the Linux iptables script just mentioned, change the permissions to make it executable, **su** to *root*, and execute it.
2. Edit the `ADCD.xxxx.TCPPARMS(GBLTDATA)` member as needed, based on the comments above.
3. Stop and restart the resolver:
 

```

(MVS console) P RESOLVER
                S RESOLVER,SUB=MSTR
      
```
4. Go to ISPF option 6 and try the **NSLOOKUP WWW.IBM.COM** command. You then see the requested address translation.

## Comments

You must find an appropriate name server. The 167.206.10.178 address shown is simply an example. An easy way to do this, assuming your Linux system is connected to the Internet, is to issue a Linux command (such as **nslookup WWW.IBM.COM**) and observe what name server is

used to resolve the name. Provide a DOMAINORIGIN name that does not obviously conflict with known names. In our simple example, this domain name is not used in any real sense.

For more resolver setup information, see *Communications Server for z/OS V1R9 TCP/IP Implementation Volume1: Base Functions, Connectivity, and Routing*, SG24-7532.

## 7.5.8 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” systems you are placing on the LAN. If your base Linux is in a virtual environment, this adds another complication. We suggest using a small, inexpensive personal router for initial zPDT LAN setup, as shown in Figure 7-7, working in a non-virtual environment.

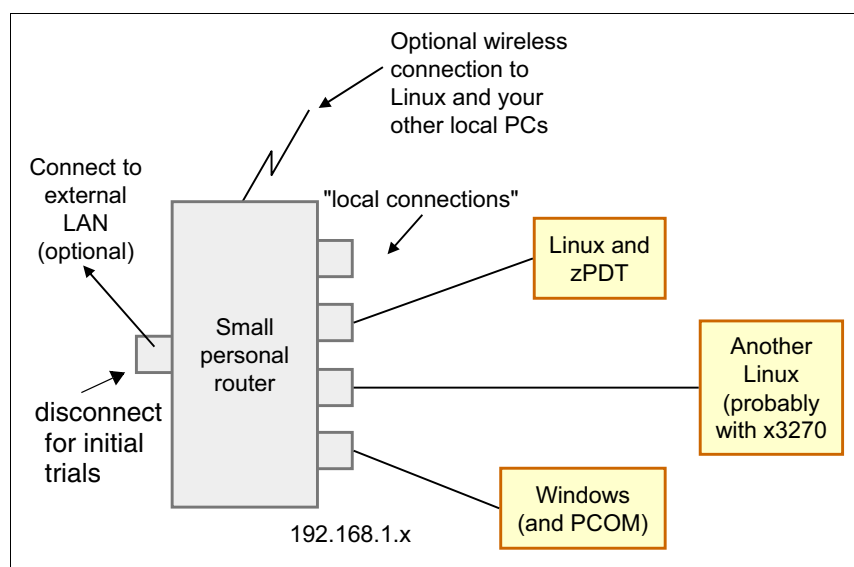


Figure 7-7 LAN debugging setup

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 and you can also explore these. (We suggest initial use with wired connections to simplify the first setup.)

After you have an environment like this working, you can transfer the operation to a larger network. Although this suggested environment is almost trivial, we often find it quite useful.

Remember that the Linux firewall (if enabled) might affect any connections. For initial debugging (especially in a private environment, as shown here) we suggest that the firewall be disabled until you verify that your basic LAN setup is working. We have observed that the Security Enhanced Linux (SEL) protection might need to be disabled or modified for some functions, such as FTP.

In most cases the zPDT user has a single network Ethernet interface available in his workplace, connected to a router somewhere external to the user. This external network interface typically expects a DHCP client, and this presents two problems:

- ▶ The System z operating system might not operate as a DHCP client.<sup>24</sup> That is, it may want a fixed IP address. In general, network-connected users do not have fixed IP addresses.
- ▶ The zPDT machine might have multiple LAN adapters, requiring 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 fixed IP addresses (provided by the NAT router).<sup>25</sup> These are typically in the 192.168.1.xxx range. The router, in turn, works with addresses provided by your external network.

Using this, 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 additional PCs to the router (which supplied its own range of DHCP addresses, if requested). The additional PCs can connect to aws3270 (using the Linux IP address and port 3270) or to OSA (using the IP address assigned, specified in the z/OS TCP/IP PROFILE).

Most routers can be configured to pass incoming port connections to specific local IP addresses. This 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 described in 7.5.3, “Scenario 3” on page 124 and in 7.5.7, “z/OS resolver” on page 130 were used with a local router and the iptables script described with the scenario.

## 7.5.9 Investigating lan problems

We have observed that problems often can be traced to external elements, such as incorrect routing specifications, external lan performance, incorrect routing for multiple lan adapters, and so forth. These are not zPDT issues and your zPDT provider cannot be expected to debug such problems.

Before asking for help from your zPDT provider you should attempt to isolate the problem. Is it really zPDT or is it due to an external element? One way to do this is to create a small private network, as discussed here (without the optional external lan connection). Try this with a basic Linux and 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.

## 7.6 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 a retransmission). This might not be noticed unless careful measurements or comparisons are made. We believe this problem is resolved by including the `sysctl` parameter that is now recommended:

```
net.core.rmem_max=1048576
```

- ▶ Linux attempts to offload various low-level LAN interface functions to the NIC adapter. This might not be compatible with the awsOSA implementation and can result in drastic

<sup>24</sup> The Linux for System z operating system might provide a DHCP client interface. The “traditional” IBM z System operating systems do not function as a DHCP client.

<sup>25</sup> The router might also function as a DHCP server, providing DHCP addresses in a portion of its address range.

performance degradations. We note that more recent Linux distributions (such as SUSE LEAP 15 or CentOS 7) *might* not experience these problems.

IBM does not provide performance specifications for zPDT OSA. When working correctly, informal observation indicates that FTP throughput might be in the 5 - 20 MB/s<sup>26</sup> (megabytes/second) range, assuming an unconstrained network in a dedicated environment. If your performance is very much worse than this, consider experimenting with the **ethtool** commands described here.

- ▶ One or more of the following commands, intended to disable the Linux offloading of IP functions, might improve the situation:<sup>27</sup>

```
# 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 VLANs)

# ethtool -k eth0             (display status of 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.<sup>28</sup> One user reported the following combination most effective for his 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.

## 7.6.1 Jumbo frames

**Important:** Not all Linux distributions support jumbo frames.

A Linux jumbo frame is a LAN frame larger than 1500 bytes. Starting with GA8, zPDT supports jumbo frames up to 9000 bytes,<sup>29</sup> 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 z operating systems) should be set to the desired value, such as 8992.
- ▶ You must also 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). Your Linux distribution may vary slightly, but the following illustrates where we made the necessary change:

<sup>26</sup> This represents z/OS FTP transfer rates to a base Linux system (on a separate PC) working over a local, unconstrained network.

<sup>27</sup> Some of these ethtool settings might not be effective for your specific adapter, but they do no harm.

<sup>28</sup> We found that Ubuntu accepted only the gso and gro changes.

<sup>29</sup> For reasons beyond the scope of this chapter, the MTU size for TCP/IP definitions is sometimes set to eight bytes less than the true MTU size; 1492 instead of 1500, 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 used.

- SUSE

Go to `/etc/sysconfig/network` and look for a file name that reflects the interface you are using; for example, `ifcfg-eth0`. Edit this file and look for a line such as:

```
MTU=''
```

Change this to `MTU='8992'`

- Red Hat

Go to `/etc/sysconfig/network-scripts` and make a similar change as shown for SUSE. If an MTU line is not present, add it. The *single quote* marks might not be needed.

- Ubuntu

We suggest using the **ifconfig xxxx mtu 8992** command described below.

A temporary change for any Linux (until Linux is rebooted) can be made 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 complete LAN connection,<sup>30</sup> between your system and whatever systems you connect with, must be capable of handling jumbo frames, otherwise the use of jumbo frames might *reduce* performance. You can test a connection for the use of jumbo frames using the following command:

```
$ ping -M do -c 3 -s 8900 my.remote.pal.com  (use your target url, of course)
$ 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 the **ping** to be sent three times. (The count number is arbitrary.)

The `-s 8900` causes **ping** to add 8900 bytes of padding to the normal ping packet.<sup>31</sup>

If the **ping** fails (typically by hanging until it times out) then your connection does not support jumbo frames. As best we understand, generally available Internet connections do not support jumbo frames.

The examples in this book use the non-jumbo MTU size of 1492 because the LAN environments capable of handling jumbo frames are not routinely available.

## 7.7 Local connections to z/OS TCPIP

If you elect to install the tunnel connection as described earlier you can connect from the base Linux to z/OS by both Telnet<sup>32</sup> (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 &           (to start a TN3270 session via z/OS TCP/IP)
$ telnet 10.1.1.2 1023       (line-mode Telnet session via z/OS TCP/IP)
```

The 1023 parameter in the **telnet** command specifies the port number that the AD-CD UNIX System Services uses for Telnet connections. This port number (1023) is not standard, and probably applies only to the AD-CD z/OS system. We have also noticed that **inet** is not

<sup>30</sup> This includes all the routers and switches between your system and the target system.

<sup>31</sup> 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.

<sup>32</sup> We use “telnet” generically, and include SSH in this category.

automatically started on some AD-CD z/OS systems; this can be corrected with the command `s inetd` on the z/OS operator console.<sup>33</sup>

## 7.8 Choices

Which 3270 connection mode is better? If only simple 3270 connections are needed (and not more than 32 sessions are needed), then the use of basic `aws3274` connections (shown in scenario 1) is better. This is simpler to set up and does not require OSA or z/OS TCP/IP to be configured or started.

## 7.9 Automatic configuration

We discussed the use of the `iptables` command for implementing *nat* functions, and the `ethtool` command for improving LAN performance. To avoid entering these commands every time Linux is rebooted, we added them to *root*'s crontab, resulting 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
```

The last two lines are created during zPDT installation and must not be altered. The `iptables` and `ethtool` commands have been described earlier. The “eth0” operand in many of the lines is the name of our ethernet port; the name might be different on your system. The “3270” operand is the port number for `aws3274` that is specified in our devmap; you might use a different port. Our example was used with openSUSE; you might need adjustments for other Linux distributions.

The effect of this table is that, after booting Linux (with the firewall enabled), starting zPDT, and IPLing z/OS, we can connect z/OS to external internet sites (due to the *nat* operation) and experience very reasonable ethernet transfer rates. (We have noticed that the *gro* function is sometimes turned back on automatically; we do not know why this happens.)

If you are not familiar with crontab customization, please do a little research before attempting changes to it.

### Alternate method

Instead of using cron to automatically issue `ethtool` commands for the base Linux you could use a slightly more sophisticated `systemd` method:<sup>34</sup>

- ▶ Working as *root*, create file `/etc/rc.d/zPDTNetwork.sh`:

```
#!/bin/bash
# zPDT OSA performance improvement
#
```

<sup>33</sup> If appropriate, add this command to the VTAMAPPL script you are using.

<sup>34</sup> This technique was suggested by Sebastian Welton. It assumes your base Linux uses `systemd` functions.

```

ethtool -K eth0 rx off          (Alter "eth0" to match your system)
ethtool -K eth0 gso off
ethtool -K eth0 gro off
ethtool -K eth0 rxvlan off

```

- Make the file executable:

```
# chmod u+x /etc/rc.d/zPDTNetwork.sh
```

- Working as *root*, create file */etc/systemd/system/zPDTNetwork.service*:

```

# /etc/systemd/system/zPDTNetwork.service
# Run the following script after booting
[Unit]
Description=Set ethtool for OSA adapters

[Service]
Type=oneshot
ExecStart=/bin/sh -c "/etc/rc.d/zPDTNetwork.sh"

[Service]
WantedBy=multi-user.target

```

Use the standard `systemctl` commands to control the function:

```

# systemctl start zPDTNetwork.service      (start now; not persistent)
# systemctl status zPDTNetwork.service     (current status)
# systemctl enable zPDTNetwork.service     (autostart after boot)
# systemctl disable zPDTNetwork.service    (no autostart at boot)

```

**Important:** Do not blindly copy our examples. You must understand your networking environment and your networking goals and adjust our examples accordingly. The `iptables` and `firewall-cmd` commands and `nat` functions discussed here might not be appropriate for your environment.

## 7.10 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 = address, nn = number to display
D M=DEV(dddd)           provides path status
D M=CHP                 display all CHPIDs defined to z/OS
D IOS,MIH               display current MIH values
SETIOS MIH,DEV=E201,TIME=00:30  example of setting MIH

```

- TSO commands:

```

NETSTAT DEV             display all devices and links
NETSTAT HOME            display home address
NETSTAT GATE            display gateway addresses
NETSTAT CONN            display connection status
TRACERTE ipaddress
PING ipaddress
(Issue ALLOC DD(SYSTCPT) DA(*) before TRACERTE or PING for more data.)

```

- VTAM commands:

```
V NET,ACT,ID=LCL701      vary local 3270 active to VTAM
```

<b>D NET,MAJNODES</b>	<i>display major nodes</i>
<b>D NET,ID=xxxxxx,E</b>	<i>display information about specific node</i>
<b>D NET,TRL</b>	<i>list the TRLEs</i>
<b>D NET,TRL,TRLE=OSATRL1E</b>	<i>data about specific TRLE</i>
<b>V NET,ID=OSATRL,ACT</b>	<i>activate a major node</i>
<b>V NET,ID=OSATRL,INACT</b>	
<b>V NET,ID=ISTTRL,ACT,UPDATE=ALL</b>	<i>remove inactive TRLEs from TRL list</i>

Note that the name LCL701 in the sample **V NET** command is the VTAM name of the terminal. This name is *not* related to the LUName specified in the zPDT devmap. A 3270 session has both an aws3274 LUName (specified in the 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 (used to link a TN3270e session to an aws3274 definition) is not necessarily the same LUName associated with a VTAM operation.

Also, note that zPDT does not support the VMAC function from z/OS. The only virtual mac supported is generated on z/VM with the layer-2 vswitch.

## 7.11 More complete 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 elected 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, 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 standar 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

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 AD-CD z/OS systems (up to the time of writing) use the DEVICE, LINK, HOME operands in the TCPIP PROFILE definition. The same are used in this book in order to match the AD-CD systems. You can use the newer INTERFACE parameters if you chose, for example:

**OLD**

**NEW**

```

DEVICE PORTA MPCIPA
LINK ETH1 IPAQENET PORTA
HOME 10.1.1.2 ETH1

```

```

....
START PORTA

```

```

INTERFACE ETH1
DEFINE IPAQENET
IPADDR 10.1.1.2/24
PORTNAME PORTA

```

```

START ETH1

```

## 7.12 VLAN usage

z/PDT OSA emulation supports VLAN usage provided the underlying Linux NIC card or NIC driver do not impose their own VLAN control. VLAN works properly in the systems IBM uses for tests, but may not work in all systems. Unfortunately, documentation at this level of detail may be difficult to find for some NIC adapters and drivers.

## 7.13 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 indicated in the message (10.1.1.1) is associated with a tunnel (tap) device in typical zPDT operation and 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`.

## | 7.14 NFS and SMB

There is no explicit zPDT support for NFS or SMB-mounted DASD,<sup>35</sup> although NFS-mounted DASD has been used during zPDT development. SMB has not been used and the usefulness of SMB with zPDT is unknown.

In principle, the use of LAN-accessed files is transparent to 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* CCWs. This locking protects some z/OS metadata (such as VTOC and catalog updates) but does not, in general, protect shared data.

## 7.15 Conflicts with z/VM

The z/VM 6.4 AD-CD has OSA device numbers (addresses) 400-402 defined for use by z/VM. This might conflict with z/OS usage when z/OS is run under z/VM, as with a parallel sysplex configuration. This conflict can be resolved as follows:

---

<sup>35</sup> SMB is also know as SAMBA.

- Simply change the OSA addresses used for z/OS. Addresses in the range 400-40F are available in the AD-CD z/OS systems.
- Or:
- See “Simple network adjustments” on page 188 for a brief description of VM TCPIP adjustments.

## 7.16 Secure (TLS) connection to z/OS

The x3270 terminal emulator can provide a secure (encrypted) connection with z/OS when using the z/OS TCP/IP interface.<sup>36</sup> Other 3270 emulators can provide the same service, but the operational details might differ. The following material was contributed by Marc Van der Meer and Paul Mattes.

### AD-CD z/OS 2.4

The AD-CD z/OS 2.4 system (available in November 2019) is configured to accept a secure connection from x3270 through z/OS TCPIP. Note that this 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
```

This example uses the 10.1.1.2 IP address that is used for “tunnel” connections referenced throughout this book. You would obviously adjust this to match whatever IP address is used by z/OS. The 2023 port number is configured in the AD-CD system

### Secure connection configuration

This section describes steps that might be used to provide a secure connection capability for a z/OS system that is not already configured for it.

The basic mechanism requires TN3270<sup>37</sup> 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. The client then uses the public key to encrypt the session setup details, including the session key to be used. This provides an encrypted 3270 session, but the client cannot be certain it is connected to the correct host.

The client can, optionally, verify that the name in the certificate presented by the host matches the expected name. By default, x3270 3.6 and later verify host certificates; earlier releases do not do this by default. Additional x3270 parameters can be used to force verification (for x3270 releases earlier than 3.6) or to disable verification (for x3270 3.6 and later) and to override the hostname being checked. At the time of writing, we have 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 host name, if desired. This is not required.
2. Create the necessary certificates, using the RACDCERT utility.
3. Update parameters in the TN3270 configuration used by z/OS TCPIP.
4. Export the CA certificate to the system(s) that will provide the x3270 sessions. This could be your base Linux and/or an external Linux system. This step is needed only if host verification is done.

<sup>36</sup> This material does not apply to 3270 connections via the aws3274 device manager.

<sup>37</sup> TN3270 is the z/OS started task that provides 3270 operation through TCP/IP/

5. Update Linux hostname resolution files, if needed.
6. Start x3270 with the appropriate parameters.

### ***Change the z/OS host name***

Recent AD-CD z/OS systems have S0W1.DAL-EBIS.IHOST.COM as the default host name. You can continue to use this or you can change it.<sup>38</sup> (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 would not be recognized by a “real” internet name server. We changed the z/OS hostname by editing the following:<sup>39</sup>

- ▶ AD-CD.Z23D.TCPPARMS(TCPDATA) -- change the DOMAINORIGIN parameter to your selection. We changed it to OGDEN.IBM. We left the HOSTNAME statement as S0W1.
- ▶ AD-CD.Z23D.TCPPARMS(GBLIPNOD) was changed as follows. (We assume this statement is primarily for documentation and that you might have more statements in this member.):  

```
10.1.1.2 S0W1.OGDEN.IBM
```
- ▶ AD-CD.Z23D.TCPPARMS(GBLTDATA) has the same structure as the TCPDATA member and we made the same changes to it.
- ▶ While not related to the secure x3270 sessions, we also changed the full hostname in the IZUPRMxx members in PARMLIB.

Stop and restart z/OS TCPIP and then try the OMVS **hostname** command to verify your changes.

### ***Create Certificates***

At the time of writing, the AD-CD z/OS system lets the TN3270 started task default to userid START1.<sup>40</sup> Create and run the following job, changing START1 and S0W1.OGDEN.IBM as required for your system. You must have RACF SPECIAL authority or the appropriate PERMITs.

```
//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')) +
```

<sup>38</sup> If your z/OS is directly connected to the internet with a routable IP address and you wish to use a domain name that will correctly route to your system then you should work with your network people to create an appropriate host name.

<sup>39</sup> Some of the dataset names mentioned below have Z23D as a middle qualifier in the name. This qualifier changes with different z/OS AD-CD releases and you must adjust the instructions accordingly.

<sup>40</sup> You can verify this by examining the TN3270 JES log, using SDSF.

```

NOTAFTER(DATE(2029/12/31)) SIZE(2048)
SETOPTS RACLIST(DIGTCERT) REFRESH

RACDCERT ADDRING(TN3270TLString) ID(START1)
SETOPTS RACLIST(DIGTCERT) REFRESH

RACDCERT CONNECT(CERTAUTH LABEL('TN3270_signerCA') +
RING(TN3270TLString) CERTAUTH) ID(START1)
SETOPTS RACLIST(DIGTCERT) REFRESH

RACDCERT CONNECT(ID(START1) LABEL('TN3270TLS') +
RING(TN3270TLString) DEFAULT) ID(START1)
SETOPTS RACLIST(DIGTCERT) REFRESH

RACDCERT EXPORT(LABEL('TN3270_signerCA')) CERTAUTH +
DSN('OGDEN.CERT.B64') FORMAT(CERTB64)
/*

```

A few comments about this job:

- ▶ The first RACDCERT statement creates a self-signed *Certificate Authority* (CA) certificate that is stored in the RACF database. The SUBJECTSDN is arbitrary; a name must be specified, but is not used for any purpose in our example.
- ▶ The ID(START1) parameter is the z/OS userid associated with the TN3270 address space.
- ▶ The second RACDCERT statement creates a certificate, signed by the CA we just created, that will be used by TN3270 to set up the encrypted sessions. The 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, we strongly suggest that you 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 we just 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, potentially, to other Linux systems that might be used to connect to our z/OS. The dataset 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 we used for starting zPDT. The *B64* suffix name (for the z/OS dataset) and the *pem* suffix (for the Linux file) follow the standard naming convention for such files.

### **Update local name resolution**

On our base Linux we edited */etc/hosts* to include the lines:

```

127.0.0.1    localhost
10.1.1.2     SOW1.OGDEN.IBM

```

### **Update TN3270 parameters**

We decided to use TCP port 2023 for secure TN3270 sessions. This is not quite arbitrary because we observed this port was partly set up this way in some AD-CD z/OS systems. We need to inspect and update a few TN3270 parameters for port 2023. The relevant parameters are in AD-CD.Z23D.TCPPARMS, starting in member TN3270. Examining this member, we saw that it *included* members PORT0023 and PORT2023. Using this AD-CD 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 TN3270TLString      ;this is the keyring you created
  ClientAuth NONE                 ;Server side only authentication
  NOSSLV2
  NOSSLV3                         ;Force TLS
  Encryption
    SSL_AES_256_SHA               ;Strong one!
  EndEncryption
  SSLtimeout 5
EndTelnetParms
```

We then used the z/OS commands **P TN3270** and then **S TN3270** to restart TN3270.

### ***Make the connection***

The x3270 program uses *prepends* (a leading character) on some operands. Note the L: and Y: prepends in the following examples. The L: prepend indicates that a secure session is requested and the Y: prepend 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:S0W1.OGDEN.IBM:2023
  - x3270 L:10.1.1.2:2023 (where 10.1.1.2 is the IP address of z/OS TCP/IP)
- ▶ With host certificate validation
  - x3270 -verifycert -cafile cert.pem L:S0W1.OGDEN.IBM:2023
  - x3270 -verifycert -cafile cert.pem -accepthostname DNS:S0W1.OGDEN.IBM L:10.1.1.2:2023

With x3270 3.6 or later:

- No host certificate validation
  - x3270 L:Y:S0W1.OGDEN.IBM:2023
  - x3270 L:Y:10.1.1.2:2023
- With host certificate validation
  - x3270 -cafile cert.pem L:S0W1.OGDEN.IBM:2023
  - x3270 -cafile cert.pem -accepthostname DNS:S0W1.OGDEN.IBM L:10.1.1.2:2023

These examples assume that file cert.pem is in the current Linux directory; otherwise a full Linux path name is needed.

### ***Additional notes***

The *-accepthostname* option is used if you want host certificate validation and if SUBJECTSDN CN attribute in the host certificate does not match the server's hostname:

```
-accepthostname DNS:S0W1.OGDEN.IBM (note the DNS prepend)
```

The “L:” prepend available with x3270, as in **x3270 L:10.1.1.2** can be confusing. The best explanation is this: if the host is set up for a secure connection, use the L: prepend, otherwise do not. The use is optional, but it speeds the initial session negotiations.

If the host is set up for secure connections and if you do not specify a certificate (using the *-cafile* parameter, for example) and if you are using host verification, then (1) the host will

begin a secure connection, and (2) x3270 will attempt to use the root CA certificates built into the base Linux system.

Should every user be on the latest level of x3270? The realistic answer is: this is not likely at present. The complete x3270 suite is rather complex, containing more than the basic x3270 terminal emulator discussed here. 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 and these “builds” 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 appropriate for your Linux distribution, you obtain whatever release was built for that particular Linux. You can build your own x3270 from source,<sup>41</sup> but this is somewhat complex.

### ***Rework 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(TN3270TLString) ID(START1)
RACDCERT LIST ID(START1)
RACDCERT DELRING(TN3270TLString) ID(START1)
RACDCERT DELETE(LABEL('TN3270_signerCA')) CERTAUTH
RACDCERT DELETE(LABEL('TN3270TLS')) ID(START1)
SETROPTS RACLIST(DIGTCERT) REFRESH
```

Notice that some names are inclosed in quotes and some are not inclosed. In these examples the ID value and the certificate and ring names are, of course, variable to match your definitions.

---

<sup>41</sup> A little web searching will find the source files.



## zPDT licenses

To use zPDT you must have a machine-readable zPDT license and an appropriate System z serial number for your zPDT z System. If you wish to install a z/OS AD-CD system, you also need a different license to decrypt the distributed IPL volumes. The licenses are provided through a USB *token* or through a remote *license server*. The serial number is provided in similar ways. See Figure 2-1 on page 10 for examples of zPDT USB tokens.

The installation and use of these licenses, servers, and serial numbers can appear complex because there are a variety of paths and techniques that might be used. The complexity is lessened if you take a few minutes to understand the basic concepts involved.

### 8.1 Basic Concepts

The most basic zPDT configuration (a “local” zPDT configuration) consists of a single USB token connected to the user’s PC. The token provides the licenses (zPDT and AD-CD decryption) and also provides the z System serial number to be used, as shown in Figure 8-1.

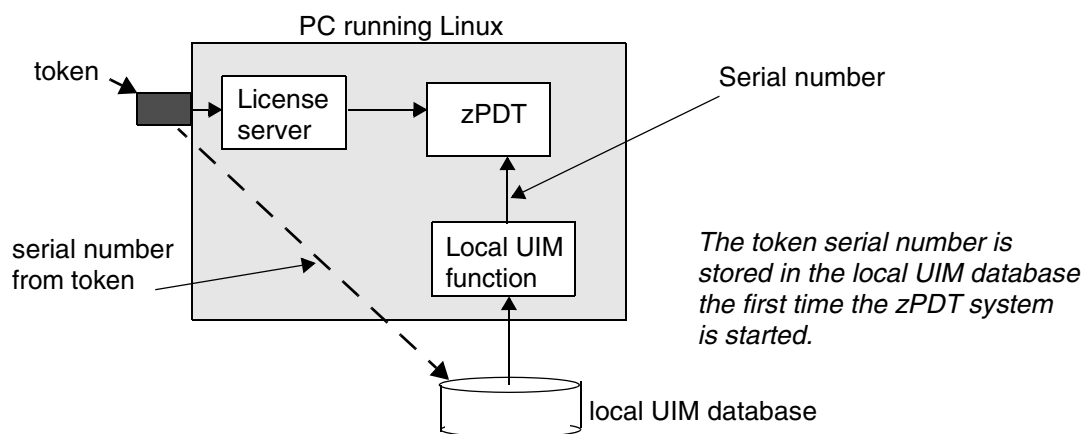


Figure 8-1 Basic zPDT licenses and serial number

System z machines have unique serial numbers, allowing software to identify the machine and LPAR, and the zPDT function that manages serial numbers is known as UIM. UIM stands for Unique Identifier Manager. A System z serial number might be important for some users, although many zPDT users simply accept whatever number zPDT assigns and do not worry about it.<sup>1</sup> In this basic, local configuration the zPDT license server and the UIM function are both “part of zPDT” and are not visible as separate elements. We illustrate them separately to emphasize the functions they perform.

The UIM data is used only when zPDT is started. The zPDT license is accessed every few minutes while zPDT is operational. Please note that multiple zPDT licenses are required if zPDT is to use multiple CPs. A typical token might contain three zPDT licenses and one AD-CD license.<sup>2</sup>

The general concept for remote license servers is shown in Figure 8-2. The figure also illustrates that multiple instances of zPDT can be used in a Linux client system.

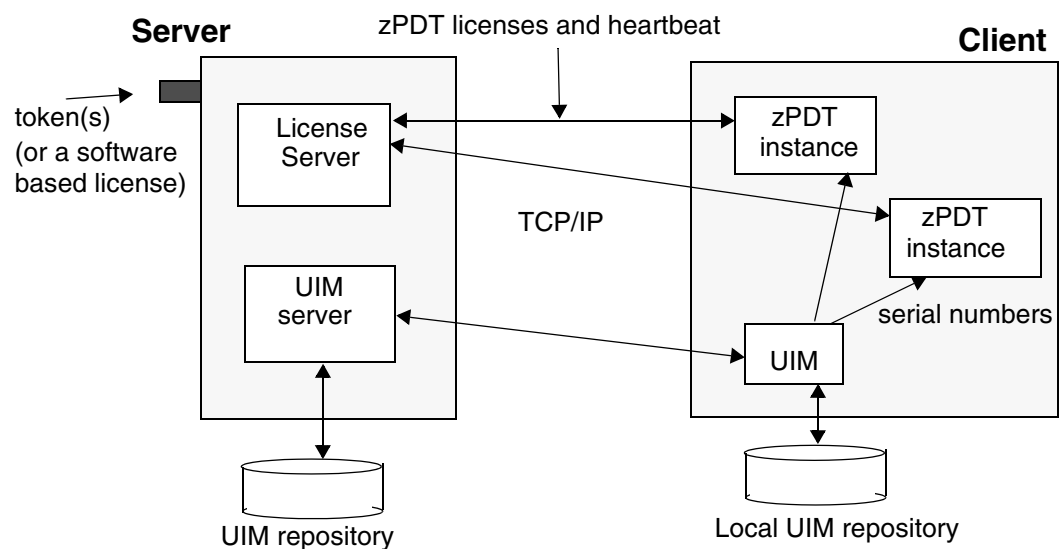


Figure 8-2 General concept of remote license and UIM servers

The obvious difference between a “local” zPDT system and a remote server configuration is that the local machine(s) (the clients) do not have a token installed. The license server can provide licenses for multiple clients on multiple PCs, limited only by the number of zPDT licenses it has available to provide to clients. The *heartbeat* is a message between zPDT instances and the zPDT license that, among other things, contains a time stamp.

The network connection between the server and client(s) is assumed to be reliable. We have seen rare cases where an unusually erratic connection (such as with a high noise level or very frequent disconnects) has caused zPDT to hang.

### 8.1.1 Types of tokens and licenses

There are multiple types of zPDT licenses, tokens, and license servers, and this can be confusing. There are two general categories of zPDT users:

<sup>1</sup> Some operating systems verify that the “IPLed” machine has the same serial number as the machine that last used that copy of the operating system and may react differently if there is a mismatch. Some software products are licensed by machine serial number, making it important for users of these products.

<sup>2</sup> The AD-CD license is used only to decrypt the z/OS IPL volumes when installing them. It is not needed for routine zPDT or z/OS use. There is seldom a need for more than one AD-CD license.

- ▶ Independent Software Vendors (ISVs).
- ▶ Commercial customers (zD&T, formerly known as RD&T).

The license control mechanisms (tokens and “software-only” licenses) are from Gemalto N.V., under the general product name of SafeNet. There are two SafeNet product families which we refer to as Gen1 and Gen2:<sup>3</sup>

- ▶ Gen1 tokens have been used since zPDT first became available and comprise both 1090 and 1091 tokens for ISV and zD&T customers. Gen1 tokens are an older product family.
- ▶ Gen2 licenses are newer and are available in both hardware (token) form and a software-only (no token needed) form. We use the term Gen2-SL for the software-only version and Gen2-HL for the hardware (token) version when we need to distinguish the two options. The Gen2 tokens are available in three different physical formats. Gen2 tokens (and software) will eventually supplant the Gen1 family. At the time of writing, zPDT uses Gen2-SL licenses only for zD&T customers and is not yet using Gen2-HL tokens. However, the zPDT GA7 and later software supports all forms of Gen1, Gen2-SL, and Gen2-HL.

There are several types of tokens or equivalents available:

- ▶ Gen1 token type 1090 is used by ISVs (and also by many IBM employees). These tokens have 1, 2, or 3 zPDT licenses. The token can be used in a local configuration or in remote license servers. The zPDT licenses in 1090 tokens allow the use of z/VM and Coupling Facilities.
- ▶ Gen1 token type 1091 is used by zD&T customers. The typical 1091 tokens have 1, 2, or 3 zPDT licenses, but higher capacity versions are available with up to 100+ licenses. The tokens can be used in a local configuration or in remote license servers. An optional feature of the zD&T license allows the use of z/VM and Coupling Facilities.
- ▶ Gen2-SL licenses are software-only (no token) licenses available for use only on remote servers. They cannot be used in a “local” configuration. At the time of writing, Gen2-SL is available only for zD&T customers.
- ▶ Gen2-HL tokens can contain either ISV or zD&T licenses, depending on how they are initialized by the zPDT provider. The 1090 and 1091 terms are not used for these tokens.

All the tokens (and the Gen2-SL equivalent) normally contain AD-CD licenses. Multiple tokens (of the same type) can be used in both local systems and remote servers to provide more licenses. A single zPDT instance cannot use more than eight zPDT licenses (corresponding to eight CPs.) Regardless of the type of token (or software license) used, an operational zPDT system identifies itself as an IBM type 1090 system.

A table summarizing tokens and licenses is included in “Introduction” on page 1.

## 8.2 Using a local zPDT system

Preparing to use a local zPDT system (that is, one with a token connected to a USB port and not connected to a remote zPDT server) depends on the type of token involved. Key points are these:

- ▶ If you are using a Gen1 token (a 1090 or 1091 token) simply connect it to a USB port. The token must have been activated with a current license. If the token does not contain a

<sup>3</sup> These are informal names we use. Some earlier zPDT documentation referred to Gen1 tokens as SHK tokens and Gen2 licenses as LDK licenses. See Gemalto materials on the Web for more formal product information.

current license see “Gen1 token activation and renewal” on page 167 for more information.

- ▶ If you are using a Gen2 hardware token, you must install the Gen2 client software. This is described in “Gen2 client configuration” on page 153. After this is installed, simply connect the Gen2 token to a USB port. The Gen2 token must contain licenses appropriate to your zPDT package; that is, licenses for zD&T operation or ISV operation.
- ▶ If you are always using the same single token, this is all that is needed. zPDT constructs a z System 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, you should read “UIM usage details” on page 150.

With this basic operation you can ignore the remainder of this chapter.

## 8.3 UIM usage details

Each zPDT instance is assigned a unique serial number. Every zPDT instance<sup>4</sup> has an LPAR ID assigned to it.<sup>5</sup> The combination of serial number and LPAR ID becomes part of the CPUID. The CPUID is the information provided by the z System instruction Store CPU ID (STIDP).

The rules for using a zPDT *license* are straight-forward. The rules for zPDT *serial numbers* are more complex. The goal is to always have the same unique serial number for a given zPDT instance.<sup>6</sup> The following general rules are used to determine the z System serial number for a zPDT instance. The term *UIM serial number*<sup>7</sup> means a serial number generated and assigned by a UIM server. Important details include the following:

- ▶ If a single local token is used (and no previous serial has been assigned, and the RANDOM option of **clientconfig** has not been selected):
  - The first zPDT startup will take the z System serial number from the token. This serial number is then written in the local UIM database.
  - Subsequent zPDT startups use the same token.
    - Or, if a different token is used, the **uimreset -1** command must be issued first (before zPDT is started). This erases the existing serial number in the local UIM database, allowing a new token (with a different serial number) to be used.
    - Or, the RANDOM parameter may be specified by using the **clientconfig** command. This 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 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 if a *UIM serial number* is present in the local UIM database (due to a previous connection to a UIM server or the use of the RANDOM option) then the UIM serial number is used and the local token serial number is ignored.

<sup>4</sup> The “instance” terminology is typically used when multiple concurrent zPDT copies (“instances”) used on a single base Linux.

<sup>5</sup> This is not the same as the LPAR *name*. The LPAR name is the same as the Linux userid that started the zPDT instance, or a system\_name from the devmap. zPDT instances have some of the characteristics of an LPAR, but full LPAR functionality is not provided by zPDT.

<sup>6</sup> Remember that multiple zPDT instances can be used on a single Linux base.

<sup>7</sup> The term *random serial number* is also used for serial numbers 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 and indicates the serial is not related to a specific token serial number. You, a client user, cannot create the “random” number.

(The local token still supplies the 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 zPDT licenses. In this case the serial number from the first token is retained. You do not use the RANDOM option in this case.
- ▶ 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.<sup>8</sup>
  - 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 may use the `uimreset -1` 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 take actions to allow a serial number change.<sup>9</sup> A user cannot assign an arbitrary serial number; the serial numbers are generated by UIM or taken from a token.

**Important:** zPDT does not allow user-assigned z System CPU serial numbers.

## 8.4 General zPDT client and server details

A license or UIM *server* is accessed (via TCP/IP) by a *client* PC running zPDT and the zPDT operational license is supplied this way.<sup>10</sup> 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 zPDT is operational on the client. Likewise, the client machine has access to a UIM server that manages consistent serial numbers for the z System CPs.

All zPDT systems have remote client functionality but, by default, it is not configured for remote operation. If a remote client function is configured, zPDT attempts to connect to remote servers to obtain a zPDT license and serial number (if one is not already set in the local UIM database).

The owner of the client machine must do some minor configuration work to enable clients to use remote license servers and UIM servers. Before enabling client access to a remote server the server networking environment (IP address, domain name, firewall controls, 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 *could* be on separate machines. A UIM server and/or a Gen1 server can be on the same machine as the client, but this is unusual and would still be considered remote servers in the context described here. All the following text assumes that the license server

<sup>8</sup> This is noted as a “random” serial number. In this case “random” simply means it is not related to a token serial number. Do not try to generalize the “random” terminology.

<sup>9</sup> This involves the `uimreset` command.

<sup>10</sup> The licenses needed to decrypt z/OS IPL volumes are also provided by the server.

and the UIM server are on the same remote machine. A Gen2-SL server cannot be present on a PC running zPDT.

Additional details include the following:

- ▶ The TCP/IP port number for a Gen1 license server is 9450, for a Gen2 license server it is 1947, and for a UIM server it is 9451. These port numbers are not configurable;<sup>11</sup> 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.
- ▶ After a zPDT instance is started (on a client) the license access must be maintained for the life of the zPDT instance. If the access is dropped, the zPDT instance stops. (If the access is recovered, zPDT starts again.)
- ▶ The servers must be identified by domain names or by IP addresses. This is easy if they have direct, fixed IP address or domain names. It is not easy if DHCP-assigned addresses or NAT functions or VLAN networks are involved. *Skilled network planning is required for any but the simplest environments.*
- ▶ If a Gen1 server is not specified, it defaults to *localhost*.
- ▶ A client machine may be changed to a stand-alone machine (with token) by changing a configuration file, and vice versa.
- ▶ In normal operation, a client machine always has the same z System serial number. This number, once assigned via a local or remote function, might not be related to any physical token number.
- ▶ Any license or UIM configuration changes should be made when zPDT is not operational.

## 8.5 Client Installation and configuration for remote servers

All Gen1 client functions (for both licenses and UIM functions) are included and installed by the zPDT installation package. Whether the remote functions are used depends on configuration options. The basic zPDT client installation process is described earlier in Chapter 5., “zPDT installation” on page 95.

### 8.5.1 Gen1 client configuration

After a normal zPDT installation, Gen1 operation with a remote server is normally configured with the **clientconfig** command.<sup>12</sup> It produces a display similar to this (assuming you are not using Gen2 licenses):

```

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)
```

<sup>11</sup> This is a change for zPDT GA7.

<sup>12</sup> You must operate as *root* to use the **clientconfig** command, or enable use of the **clientconfig\_authority** function.

Parameters are changed by simply overtyping them. Remember that configurations for two separate functions (Gen1 license server and UIM server) are specified here. The general rules are these:

- ▶ The Gen1 `ContactServer` should be `localhost` (to specify that no remote Gen1 license server is being used)<sup>13</sup> or the address (IP address or domain address) of a Gen1 license server. One of these options must be specified. The default *localhost* parameter causes zPDT to operate with a local token.
- ▶ The Gen1 `BackupServer` field is used only if there is a second Gen1 license server.
- ▶ If the Gen1 `ContactServer` is not `localhost` (that is, a remote Gen1 license server is being 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 could be `localhost` (if a UIM server is running on the client machine) or the address (IP or domain name) of a remote UIM server.
- ▶ If the Factory Reset option is set to “y”, all other parameters are ignored and the configuration file is restored to the original values shipped with zPDT.

Changes to the configuration file are not dynamic. They take effect only when zPDT is started. Note that earlier versions of `clientconfig` provided a field for a UIM server port number; this option is no longer relevant.

An alternate method for setting the `clientconfig` values for Gen1 and Gen2 licenses is described in 4.1.26, “The `clientconfig_cli` command” on page 64. This alternate method provides a command-line interface that could be used within a Linux script.

The actual server information is in 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>
```

We *strongly* suggest that you do not attempt to edit this file directly. Directly editing XML files is very prone to errors and debugging can be difficult.

## 8.5.2 Gen2 client configuration

You must complete two steps for Gen2 client operation: (1) activate the Gen2 software, and (2) configure your specific server information.

After the normal zPDT package is installed the Gen2 client can be activated.<sup>14</sup> Working as *root*, issue the command `/usr/z1090/bin/gen2_init`.

<sup>13</sup> If the GEN1 `ContactServer` field is blank, it is internally defaulted to `localhost`.

The resulting display depends on your Linux distribution, but might look like the following:

```
# /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 is started automatically when the client Linux system is booted.

Next you must tell your client where to find the server, 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 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 would be unusual.

The specification order for a UIM server is (1) the UIM ContactServer, (2) the Gen2 Server, then (3) 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 would normally leave the UIM ContactServer field blank.

If the Gen1 ContactServer field is left blank, it internally defaults to localhost.

An alternate method for setting the clientconfig values for Gen2 licenses is described in “The clientconfig\_cli command” on page 64, or the “The ldk\_server\_config command” on page 71.<sup>15</sup> These alternate methods provide a command-line interface that could be used within a Linux script.

The **query\_license** command can be used with Gen2 clients and servers and is shown in “Managing the Gen2 server” on page 157. It is useful for verifying your setup and connectivity with the license manager.

SafeNet provides a browser-based *Sentinel Admin Control Center* that could be used to configure the Gen2 functions. We *strongly* recommend you use the **clientconfig** and **serverconfig** commands rather than this browser interface.

### 8.5.3 Client UIM configuration

The configuration details are:

- 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

<sup>14</sup> 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 if the library was not already present in the Linux system. It was necessary to have a working Internet connection before starting the process to install the client or server.

<sup>15</sup> The ldk\_server\_config command is deprecated; the clientconfig\_cli is the recommended command.

assumed to be at the same IP address as the Gen1 server and uses a port number 9451. This is described in more detail in “Gen1 client configuration” on page 152.

- ▶ For a Gen2 server you can use the **clientconfig** command and set the **UIM ContactServer** and **UIM PortNumber** variables.
- ▶ The **clientconfig\_cli** line command may 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 a zPDT machine. Any user may issue this command.

```
$ uimcheck (may be used by any userid)
```

## 8.6 Server installation and configuration

Both the Gen1 license server and UIM server are included in the base zPDT package. The license server runs as a daemon and is automatically started when Linux is booted. (This is true even for local token use.)

The Gen2 license server (packaged with the standard UIM server) is *not* part of the standard zPDT package. A package with these two components is available as a separate deliverable. (As mentioned earlier, at the time of writing the Gen2-SL offering is only for zD&T customers.)

### 8.6.1 UIM server

The UIM server used with a Gen1 server is automatically installed when installing zPDT. The separate package providing a Gen2 license server also contains a UIM server. The UIM server is the same in both cases.

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 userid and places its database in the home directory of that userid. It also places small log files in the home directory. For this reason, the same Linux userid (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)
```

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 reboot. 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 userid running the UIM server; this will corrupt the file due to checksums that are within it.

If you must change the UIM server port number you should consult your zPDT provider for guidance.

## 8.6.2 Gen1 License server

The Gen1 license server is part of the standard zPDT package and is installed as if you were installing a zPDT client. It is activated by the actions of the two token “driver” components that are part of zPDT installation.

One (or more) 1090 or 1091 tokens must be installed in the license server machine before it can be used. The license server configuration file is located in:

```
/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 you would then need to restart the server:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart
```

Several security functions may be specified in the sntlconfigsrvr.xml file.

## 8.6.3 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 similar to that shown in the following command. Place this file in a convenient directory and, working as *root*, execute the file:<sup>16</sup>

```
# ./zPDT_LS-1-8.53.07.x86_64 (Use your correct file name)
```

Both the Gen2 license server and a UIM server are installed.<sup>17</sup>

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*, issue the following command:

```
# /opt/IBM/LDK/request_license
```

This creates a file 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 zPDT licensing facility (as identified by your zPDT contract). In return you will receive an “update\_zip” file containing the number and type of licenses your server can supply to clients. Receive this file into a convenient directory and install it as follows:

```
# /opt/IBM/LDK/update_license <hostname>_xxxxxx_update.zip
```

Then 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)
```

This completes the Gen2 license server installation. Remember that you also must start the UIM server on your server system. You should consider security controls for your Gen2 server as described in “Security” on page 163.

The update\_zip file that conveys zPDT licenses to the server also contains AD-CD decryption licenses that become available to the client systems.

<sup>16</sup> The file must be executable. This might require a **chmod u+x** operation. Also, the exact file name may change slightly to match newer levels of zPDT.

<sup>17</sup> The Gen2 server is installed in /opt/IBM instead of the traditional /user/z1090/bin that has been used for other zPDT modules.

**Important:** You cannot reinstall a Gen2 license file. You cannot “start all over” with it. If you need to start all over, you must run the *request\_license* function again and obtain a new *update\_license* file from your zPDT provider.

The Gen2 software license server is tied to the physical PC whose “fingerprint” was used when obtaining the licenses. It cannot be moved. It cannot be relocated in a virtual environment.

## Managing the Gen2 server

Once a Gen2 server is installed, the **serverconfig** and **query\_license** commands may be used to help manage it. The **serverconfig** command, used on the Linux system that is running the Gen2 server, produces the following display:

```
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, it 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, as described in “Security” on page 163.

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 being used. For a server, it displays the inventory of licenses available and information about all current users.

```
$ query_license
The following key is available:
HASP-SL key=id=367116869417668380 feature(s):
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 3 CPs would have 3 “logins.” In this example the server has 12 zPDT CPU licenses, and only one of these 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. You cannot simply move the server files (or the hard disk containing the files) to a different machine.

The *man* files for a Gen2 license server are located in `/opt/man/IBM/LDK/man1`. The access them you need to include this directory in the *man* path for *root*. For example:

```
export MANPATH=$MANPATH:/opt/man/IBM/LDK
```

## 8.7 General Notes

Do not run **Z1090\_token\_update** or **Z1091\_token\_update** from a client zPDT machine when using a remote license server. The utility cannot affect tokens or licenses in the remote license server, but will attempt to access a token in the local PC. You may run the utility in the Gen1 license *server*, to update the tokens in the server.<sup>18</sup>

The client UIM information is held in `/usr/z1090/uim/uimclient.db`. In unusual error situations you might be advised to delete this file. This will cause the UIM function to obtain or create a new serial number when zPDT is next started, as described in “UIM usage details” on page 150.

The administrator of a license server is responsible for ensuring 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 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 will furnish the license (or licenses) for a zPDT startup.

The license expiration date displayed by the **token** command (in a client machine) may not reflect the effective expiration date of all the active tokens in a license server. The **token** command (when 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 zPDT license was obtained from token 0x159FA (decimal 88570) and the CP serial number used by 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.)

### Starting all over

If you decide to “start all over” and reinstall your 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), zPDT will obtain the same serial number from it. If you use a remote license server and deleted any previous references there (with a **uimreset -r** command) your new zPDT installation might not have the same serial number as the previous setup. If you do not care about z System serial numbers then this is not a problem. If you do care about z System serial numbers (due to software contracts or software sensitivity) this can be a problem. The only certain way to obtain the same z System serial number is to use the same single local token without the **RANDOM** option.

You cannot “start all over” when installing a Gen2 software-only licenses.

---

<sup>18</sup> 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.

## 8.7.1 Firewalls

You (or your networking people) must manage any firewalls involved with remote servers and ensure that intermediate routers can find your server and your client.<sup>19</sup> 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 system respond to **iptables** commands, such as:

```
# 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
```

## 8.7.2 Disk and Linux changes

Changing the Linux disk (HDD) might change the identifier that is part of the identification used by UIM. You may need to reset the local serial number (**uimreset -l**) or the remote serial number (**uimreset -r**) after changing the hard disk.

Upgrading to a new Linux kernel *might* change the identification used by UIM. You may need to reset the local serial number (**uimreset -l**) or the remote serial number (**uimreset -r**). If this does not solve the problem, delete the UIM database at `/usr/z1090/uim`.

## 8.7.3 Backup servers

Backup license servers can be specified for Gen1 and Gen2 license servers. Backup servers cannot be specified for UIM servers.

## 8.7.4 Cloning zPDT

If you clone a zPDT system, you must delete the files in `/usr/z1090/uim` on the new system. This is because the UUID of the new system differs from that of the old system. zPDT will build new uim files when the new system is started.

## 8.7.5 Removing functions

All Gen1 server functions (and associated UIM) can be removed by simply removing zPDT on that server. For example, either of the following methods can be used:

```
# z1090-1-7-49.28.x86_64 --removeall      (note two dashes)
```

The Gen2 client function can be removed with this 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 (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 also automatically removes the UIM server that was associated with the Gen2 server.

---

<sup>19</sup> As we have stressed before in this book, skilled networking help may be required to implement remote access to zPDT and/or the use of remote license servers.

Do not use **rpm** commands to remove these functions; it will not remove all the necessary modules.

## 8.7.6 License expiration notification

Starting 30 days before a zPDT license expires, zPDT issues a message three times per day in the Linux command window that was used to start zPDT. (That is, the command window where the **awsstart** command was issued.) This Linux window is often not visible to zPDT users or administrators, and a second notification option is available.

If you define environmental variable **ZPDT\_EXP\_EMAIL** to specify an e-mail address, a 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 command window that starts zPDT and issued before zPDT is started. For this method to work your Linux **mail** command must be properly configured and operational. This often is not a trivial task, and varies so much that we cannot advise you 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
(ctrl-D)                                         (Use ctrl-D to end your message)
```

If this sequence is successful in sending the test message to your specified address, you have **mail** 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 zPDT on the license/UIM server (Gen1)
- ▶ Switch tokens (Gen1)

### License search order

zPDT attempts to obtain a license from a Gen2 server (if one is configured), then attempts to obtain a license from a Gen1 server (if one is configured), and lastly attempts to obtain a license from a local token. There is a considerable timeout involved in trying to access the two servers, and depending on this automatic “fall through” search is not reasonable for normal operation. The **--localtoken** option of the **awsstart** command simply “short circuits” any attempts to use remote license servers and UIM servers.

### Local to remote server

Consider zPDT systems A and B (each using a different PC for zPDT with Gen1 tokens). System A has a zPDT token with serial number 12345.

- ▶ The system A owner installs token 12345 in his PC and starts zPDT. When this is done, serial 12345 is recorded in the local system UIM database. (This assumes there was no prior conflicting information in the local UIM database and the **RANDOM** option is not used.) System A may 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 zPDT not running and working as *root*, the owner configures a client as described in “Gen1 client configuration” on page 152 or “Gen2 client configuration” on page 153.
- ▶ 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 a 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/UIM servers. It will have this serial number every time zPDT<sup>20</sup> 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/UIM servers and without the RANDOM option). At this point both A and B have the same zPDT serial number; system A obtaining its zPDT license from the license server and system B obtaining its zPDT license from the local token. There is no way to avoid this.
- ▶ If the system B owner then connects to the license/UIM servers, the UIM server sees serial 12345 in B’s local UIM database and terminates the zPDT instance because 12345 has already been assigned to system A.
- ▶ The problem is that A and B both want to use the same serial number (12345) and the UIM server has it assigned to A. There are two ways to resolve this:
  - The system B owner can issue **uimreset -l** to clear the serial number in the local UIM database. The owner can then connect to the remote servers and receive a new random serial number.
  - Or, the system A owner can issue **uimreset -r** to clear his serial number from both the local and remote UIM databases. The next time system A zPDT starts, it will request a new random serial number from the server. System B can then use the 12345 token and serial number.

## Temporarily switch from server to local

Assume a notebook zPDT system is normally used with remote license and UIM servers. You want to take the system home overnight, and the servers cannot be accessed from home. If a token is available, you can start 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, of course, have a token to supply a license. In this case the serial number 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 is done.)

## Remote server to local

Assume a system owner has been using a remote license server and UIM server. To change to routine use of a Gen1 local token, the owner should use the **clientconfig** command to change the `LicenseContactServer` value to `localhost`. 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

<sup>20</sup> To be more precise, we should say “every time this same zPDT instance is used.” Multiple zPDT instances (on the same machine) must run under different Linux userids. The serial number for each of the instances will use the “LPR” portion of the serial number to differentiate the instances.

the **uimreset -l** command was used), the serial number of the local token is placed in the local UIM database and then used by zPDT.

### Using zPDT on the license/UIM server (Gen1)

Suppose we want to run zPDT on the same machine that is running the Gen1 license server and UIM servers. In this case we use the **clientconfig** command to specify License ContactServer as `localhost` and UIM ContactServer as `localhost`. This 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 zPDT on this system the user must issue a **uimserverstart** command.

Give some thought to the Linux userid that issues the **uimserverstart** command. The same userid must always be used for this command because the UIM server database is created in the home directory of this Linux userid.

- ▶ No special setup is needed for the license server. Any zPDT system (meaning the SafeNet server that is installed with zPDT) can act as a Gen1 license server.

Such combined operation (server and client) is not possible with an Gen2 server.

### Switch tokens (Gen1)

In this case, token 12345 has been used with a newly installed zPDT system. When zPDT is first started, this serial number is written in the local UIM database. If a different token is used on a subsequent startup, the zPDT startup fails. A **uimreset -l** command is needed to remove serial 12345 from the UIM database; after that, 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 may be used; the operational serial number will be taken from the local UIM database.

The important point is that zPDT recognizes the difference between a UIM server-assigned serial number (which can be used with any token) and a locally installed serial number (taken from a local token). A locally installed serial number must match the token being used (unless the **RANDOM** option is set).

### Change from single token to multiple tokens (Gen1)

Assume you start with a single token and later, for some reason, want to randomly use one of several tokens you possess. You are not using a remote license server. To do this, take the following steps:

1. Issue a **uimreset -l** command. (This requires *root* authority.)
2. Use the **clientconfig** command and set the UIM Local Serial Random value to **Y**. (This also requires *root* authority.)
3. A random serial number is generated the first time zPDT is started.
4. Thereafter you should be able to start zPDT with any token or with multiple tokens attached. The serial number assigned in step 3 is used, regardless of which token you are currently using.

## 8.8.1 Display serial number assignments

You can display the zPDT serial number assignments by pointing your browser to your remote UIM server (<http://uimserveraddress:9451>). The display is similar to what is shown in Table 8-1; this function is not valid for a local UIM database.

Table 8-1 Processor serial numbers on zPDT UIM server `ibmsys1@xxx-510.ibm.com`

Serial	Host	UUID	Year	Day
2099	d2x2.pok.ibm.com	E6D96D01-493E-11CB-AD29-B8F42F7F8461	2016	009

## 8.8.2 Security

If the license managers are used only from a single subnet, or a well-designed 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 could provide zPDT licenses to someone completely external to your enterprise.

**Important:** License and UIM server port numbers (9450 and 9451 for Gen1, 1947 and 9451) must not be used by any other IP function on your subnet. A port number conflict can cause 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 specified or not.
- ▶ The Allowed Site Address list determines which clients may obtain zPDT licenses from this server. If the list is empty (the default) then *any* client may obtain a license from this server.
- ▶ The Blocked Site Address list specifies client addresses that may *not* 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 `sntlconfigsrv.xml` file in `/opt/safenet_sentinel/common_files/sentinel_keys_server/` and may be edited there. They can also be managed by pointing a browser to port 7002 on the machine running the SafeNet license server, for example:

`http://localhost:7002` *(if working on the server machine)*

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. We suggest using 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 using the browser interface, be certain to click the **update** button on the web page after keying updates to the lists. You must then restart the SafeNet server:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart
```

These lists provide one way to secure use of a zPDT license server. Other methods, such as restricted router interfaces or non-routable IP addresses, might be more appropriate.

## Gen2 server

The Gen2 server inspects file `/opt/IBM/LDK/rules.ini` for security rules. This is a simple text file that you manage with your favorite Linux text editor. The file should contain lines like the following:

```
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 the above example any client on subnet *192.168.1* or the subnet (or host) resolved by *my.favorite.domain* can obtain a license from the server (if any are available, of course). Everyone else is rejected.

You can change the `rules.ini` file while the server is running. To do this, issue:

```
# serverconfig -u
```

to cause the Gen2 server to reload the rules file.

## 8.8.3 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 will still be provided by a UIM server (if the client is configured for connection to the server, of course). In this case, the **uimreset -r** command may be used to remove the relevant entry from the UIM server database if that is desired. If the `/usr/z1090/uim` directory is deleted it can be recreated by the following:

```
$ su
# cd /usr/z1090
# mkdir uim
# chgrp zpdththru uim          This group name is required
# chmod 2770 uim              These permissions required
```

A UIM *server* can be reinitialized by removing everything in the `UIMserver` subdirectory in the home directory of the Linux userid that runs the UIM server. This is a drastic step, of course, and 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 may be restored to its original state (which does not reference any remote servers) by using the Factory Reset option with the **clientconfig** command.

## 8.8.4 SafeNet module restarts

There are two SafeNet functions involved with zPDT. One is the license servers (Gen1 or Gen2) that we discuss in this chapter. The other is a daemon ("token driver") that communicates with Gen1 tokens (in USB ports). After zPDT is installed, both these functions

are started automatically when Linux is booted. 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:

```
$ 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:

```
# cd /opt/safenet_sentinel/common_files/sentinel_keys_server
# ./loadserv restart                (or status or stop or start)
```

The commands to restart the Gen2 server are:

```
# systemctl restart aksusbd.service (newer Linux distributions)
# service aksusbd restart           (older Linux distributions)
```

### 8.8.5 Gen2 servers

License renewal is done by obtaining a new update file from your zPDT provider and using the following command:

Receive this file into a convenient directory and install it as follows:

```
# /opt/IBM/LDK/update_license <hostname>_xxxxxx_update.zip
```

This is the same command that was used to install the initial Gen2 license enablement file.

## 8.9 Server search

A backup Gen1 and/or Gen2 server may be specified by a client. The servers are searched for an appropriate license in the order listed. There is no coordination among multiple servers; each must have available licenses (additional tokens for Gen1 or software entitlements for Gen2-SL) in order to serve them to clients. This means that the customer installation has purchased additional licenses or has split the available licenses among multiple servers in some way.

A zPDT client searches all available license sources until it finds the license(s) it requires. Gen2 servers (if any are defined for the client) are searched first, followed by Gen1 servers (if any are defined for the client), followed by locally installed USB tokens.<sup>21</sup> 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, zPDT internally specifies *localhost* for this operand. This means that, if all other license searches fail, zPDT looks for a local Gen1 token in the client system.

Remember that a Gen2-SL license server cannot be simply shifted to another PC. Moving a Gen2-SL license server function to a different PC involves multiple interactions with your zPDT license provider to ensure that the license entitlement information is removed from the old server and that a new license entitlement update\_zip file is created for the new server.

---

<sup>21</sup> The --localtoken operand of the **awsstart** command simply shortens this search to go to USB access immediately.

## 8.10 Numbers

Consider a remote license server (Gen1 or Gen2) with five zPDT licenses it can allocate to clients. A single client could request all five licenses by coding *processors 5* in his devmap. Or five different clients could each request a single license. Or there could be a combination of clients that consume the five available licenses. When a client zPDT ends (with the **awsstop** command) the licenses used by that client are available to other clients. At any given instant no more than five zPDT client licenses, representing five CPs, can be allocated to clients.<sup>22</sup>

Over time, many client zPDT systems might connect to this remote license server, provided that not more than five licenses are allocated at any one time. Each of the many clients will have a unique serial number provided by the remote UIM server. Thus we might have a case where five licenses are available but ten 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.<sup>23</sup>

A single zPDT instance cannot have more than eight CPs,<sup>24</sup> each requiring a zPDT license.<sup>25</sup> (IBM contract conditions might have a smaller limit.) Assuming that the maximum of eight could be used, the devmap for an instance could request eight licenses from the remote server.<sup>26</sup> In our example, only five licenses are available and the client would receive all five licenses (if no one else is using any licenses). Perhaps the intention of the customer is to share his five licenses among several development systems. There is no technical way to prevent a single user (that is, a single zPDT system) from using all the licenses (up to eight, if that many are available). Management control is needed to ensure “fair” sharing of zPDT licenses in situations where a limited number of licenses are serving multiple remote clients.

## 8.11 Dedicated key server

ITC<sup>27</sup> 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 in Figure 8-3 on page 167. The zKey has a Single Board Computer, running Linux, having 4GB memory and a 64GB SSD drive. It has a display port, a USB2 port, a USB3 port, and two LAN 10/100/1000 ports. A user-provided display, mouse, and keyboard would be connected when configuring the zKey, but later operation would not have these connected.

The internal Linux (in the zKey) is Ubuntu and normal Linux commands are available for customization. The zPDT commands associated with a Gen2 software license server are available and the license installation and operation is the same as described earlier in this chapter. As an additional option, the Web-oriented tools of the zD&T Enterprise Edition can also be install in the zKey.

While the zKey offers no functional advantage over a “normal” PC used as a license server, it has practical advantages in size, operational stability, and the absence of external controls

---

<sup>22</sup> We ignore differences for speciality engines in this discussion.

<sup>23</sup> This is a technical discussion. There might be additional contractual limitations to the number of users (people) or clients (zPDT systems) involved.

<sup>24</sup> zIIPs, zAAPs, and IFLs are also counted along with CPs. The total cannot exceed eight.

<sup>25</sup> A PC running zPDT must have more cores than the number of zPDT CPs requested by the devmap. In this discussion we assume the client has sufficient cores to meet this requirement.

<sup>26</sup> Starting with zPDT GA8, zIIP processors do not require a license. However, they still count toward the maximum of eight CPs in a zPDT instance and toward the number of cores needed.

<sup>27</sup> Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 or sales@itconline.com.

(after customization) that tempt curious fingers. The intention is a “set it and forget it” appliance. For more information please contact ITC.



Figure 8-3 zKey (and uPDT Enterprise Server)

As a side note, the server in the figure (an ITC uPDT Enterprise Server) was, at the time of writing, the fastest zPDT machine we have used and might be of interest to those pushing the boundaries of zPDT performance.

## 8.12 Gen1 token activation and renewal

**Note:** At the time of writing, the material in this section (about token license renewals) applies primarily to users of 1090 tokens. The mechanisms for 1091 (zD&T) token renewals may differ; consult your zD&T supplier for details.

zPDT Gen1 tokens must be *activated* (that is, have an initial zPDT license installed). This might be already done by your zPDT supplier. zPDT token updates are needed to extend the license date (or to replace a corrupted license).

The following text is relevant to basic Gen1 token users. More complex zPDT license management topics, including Gen2 tokens<sup>28</sup> and software-only licenses, are covered in other sections of this chapter.

### 8.12.1 Overview of Gen1 token updates

ISV zPDT (1090 tokens) and zD&T (1091 tokens) have different procedures for obtaining token licenses.

- ▶ ISV zPDT (1090 tokens):
  - Use the **Z1090\_token\_update** command to generate a request (.req) file based on the token.

<sup>28</sup> Gen2 tokens are a new family of tokens. zPDT GA7 accepts these tokens, but IBM is not (at the time of writing) distributing them yet. The older tokens (“Gen1 tokens”), with 1090 and 1091 names, will continue to be supported.

- Send the .req file to your zPDT license provider. (This could be IBM Resource Link for IBM employees, or an external provider for ISVs.)
  - The zPDT provider returns, usually by e-mail, a .zip file that is installed with the **Z1090\_token\_update** command. *Do not unzip the file.*
- 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 .zip file that is installed with the **Z1091\_token\_update** command. *Do not unzip the file.*

Older zPDT releases, prior to zPDT GA5, used the **SecureUpdateUtility** command to obtain .req files or install .upw files. This utility is still provided and instructions for use appear in earlier zPDT documentation.

The installation of the licenses in the .zip file (with the **Z1090\_token\_update** or **Z1091\_token\_update** command) can take longer than you might expect -- perhaps up to a minute; be patient when using the command. Remember to unplug a token for 10 - 15 seconds after updating it. This causes Linux to read the new license that you just installed.

## Multiple tokens

Only one token can be present in a system that is updating the token and zPDT can not be active when applying the update. You can create a request file (.req) while zPDT is active, provided that 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 in order to do these activities; you cannot update a token in a remote server.

### 8.12.2 Gen1 token license update details (1090 tokens)

**Important:** The following details are primarily for IBM internal users. For other users, your zPDT supplier will probably handle these details for you.

A USB hardware key (token) is normally valid for a year from the time it was last activated.<sup>29</sup> Activation (and *lease renewal*) might be handled by your 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 8-4) attached to the USB hardware key.<sup>30</sup> Store this information in a safe place because this information is needed to activate a replacement token if replacement becomes necessary.

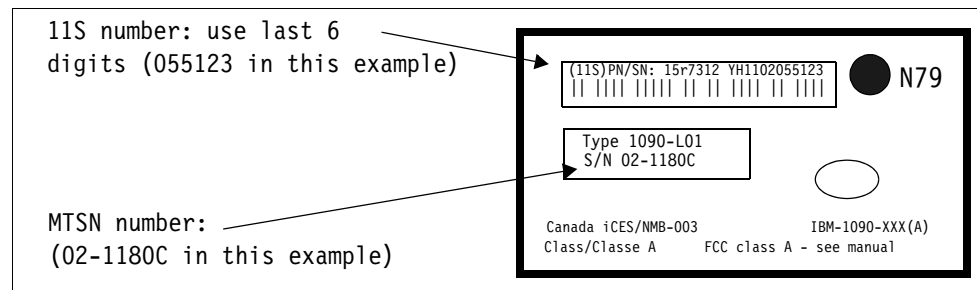


Figure 8-4 USB hardware key tag

<sup>29</sup> This was true at the time of writing. Future availability plans may manage this in a different way.

<sup>30</sup> 1091 tokens do not have this tag. They have a serial number engraved on the back, but it is not used for the process described here.

An IBM Resource Link profile (user ID) is needed.<sup>31</sup> This can be an IBM employee profile or a PWD-approved profile for other users. An IBM employee can go to the following link and follow the **Register for an IBM ID** link (in the upper right part of the page):

<http://www.ibm.com/servers/resourceLink>

After this preliminary work, the token license can be activated, renewed, or have the lease date extended (these are all essentially the same operation):

1. Connect the USB hardware key to your zPDT system, using any USB port. (You must have the zPDT software installed already.) Only one token can be connected while doing this task.
2. Working as *root*, create a request file using **Z1090\_token\_update**, or **Z1091\_token\_update**:

```
(log in with a normal zPDT userid, 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 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, navigate to **Tools** → **1090 Support** → **Date Extension** and enter the data from your hardware key tag. Use the last six 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; we suggest entering your PC serial number there. Finally, click **Submit**.
5. Resource Link creates one or two files and send them to you by e-mail. Receive the files and move them to your 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<sup>32</sup> the file to the Gen1 token:

```
# Z1090_token_update -u myrequest.zip    or
# Z1091_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 zPDT operation.

## 8.13 Summary of relevant zPDT commands and files

The following 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 path to use **clientconfig** from a non-root Linux userid.
- ▶ **gen2\_init** - installs the modules needed to work with Gen2 licenses.
- ▶ **query\_license** - displays license usage for Gen2 servers or clients.

<sup>31</sup> For various historical reasons, some IBM employees have customer Resource Link userids. These userids will not see the required links in Resource Link.

<sup>32</sup> Remember that **SecureUpdateUtility** is used with zPDT releases prior to GA5, and **Z1090\_token\_update** is used with zPDT release GA5 and later.

- ▶ **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, now deprecated.
- ▶ **SecureUpdate\_authority** - provides a path to use **SecureUpdateUtility**, **Z1090\_token\_update**, or **Z1091\_token\_update** commands from a non-root userid via the **zPDTSecureUpdate** command.
- ▶ **serverconfig** - provides controls for Gen2 license servers.
  - **serverconfig\_cli** - provides a non-interactive interface to serverconfig functions.
- ▶ **token** - displays details of the operational licenses in a client or local zPDT instance.
- ▶ **uimcheck** - displays (for a client or local zPDT system) the current UIM status.
- ▶ **uimreset** - removes the z System serial number from the local and/or remote UIM database.
- ▶ **uimserverstart** and **uimserverstop** - start and stop a remote UIM server.
- ▶ **update\_license** - installs a new license file (which was obtained from an IBM-authorized facility) containing Gen2 software licenses.
- ▶ **Z1090\_token\_update** and **Z1091\_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 presented here can help some of the key terms.

- ▶ **SafeNet**: The product line that provides the USB keys and the software that directly supports them. This includes the USB driver, the license manager, and a web interface to the license manager. The owning company is now Gemalto N.V., but the SafeNet name is used with the products described in this chapter.
- ▶ **SafeNet Sentinel Key**: The USB “token” from the SafeNet product line. This token provides zPDT license information. However, a “key” can also be a software-only function of a Gen2 server or a Gen2 hardware token.
- ▶ **Token**: Another term for a SafeNet Sentinel Key. The terms *token*, *key*, *SafeNet key*, and *Sentinel key* are used interchangeably.
- ▶ **License**: A logical function that enables one z System CP for a zPDT system. Multiple CPs require multiple licenses. The token functions (or the software-only license function of a Gen2 server) provide licenses.
- ▶ **License Monitor**: A web browser interface that displays information about Sentinel Keys and clients using them. A Gen1 monitor is accessed at port 7002 on a Linux system running a Gen1 license server, but might not be functional on recent Linux distributions. The equivalent Gen2 information can be obtained with a **query\_license** command.
- ▶ **Heartbeat**: The periodic accessing by zPDT of the license (or licenses) managed by a license server. If the heartbeat is missed, the zPDT license is revoked.
- ▶ **Time Cheat**: The Sentinel Key records the current date and time each time the key is accessed. If the Linux system clock contains a time earlier than 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 assigned by IBM. This serial number *might* be used as the basis for the z System CP serial number in some cases.
- ▶ **UIM or Unique Identifier Manager:** This is a server (or local function of zPDT) that helps maintain unique enterprise-wide z System serial numbers for 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 z System STIDP instruction. (The instance number is similar to an LPAR number on a larger z System.)
- ▶ **Serial Number:** A value in the range of 1 and 65535 (four hex digits). The serial number is assigned by the UIM function to the base Linux and used by zPDT to provide the z System serial number.
- ▶ **Instance Number:** A number in the range of 1 and 255 assigned to each zPDT instance on a base Linux machine. Each zPDT instance must operate under a different Linux userid and the instance number is assigned to the userid. The instance number is used in the same manner as the LPAR number on a larger z System.
- ▶ **UIM Database:** A file containing UIM information. The files are not directly editable. There are two types of databases. One exists in every Linux zPDT machine, and the other exists in a UIM server (if this is used). The local database (on a zPDT client) is at this location:  
`/usr/z1090/uim/uimclient.db`
- ▶ **Random serial number:** This is 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 zPDT) with a license from any token. (Do not take the “random” word too literally; in this case it means that tokens with serial numbers other than the one used to set the UIM serial number may be used. It does not mean you can select a random number.)
- ▶ **Rational License Server:** This has no relation to zPDT license servers. It provides controlled access to multiple IBM software products and might be used in conjunction with zPDT license servers.





## Other System z Operating Systems

This document is primarily concerned with z/OS. However, other System z Operating Systems may be used under zPDT.

### 9.1 z/VSE

z/VSE is available (for licensed users) as an AD-CD download for zPDT. We suggest using this download instead of installing z/VSE from the standard DVD distribution media. Consult your z/PDT provider for information about licensing and downloading.

At the time of writing, z/VSE was not available for 1091 users.

### 9.2 Linux for z Systems

Linux for z Systems is not an IBM product and is not distributed by IBM. Several Linux products and distributions can be considered under the generic name of “Linux for System z” or “Linux for z Systems.” There is no equivalent of an AD-CD available for Linux for z Systems.

Red Hat Enterprise Linux for System z, SUSE Linux Enterprise Server for System z, and Ubuntu (for z systems) have been used with the current release of zPDT.

Various installation procedures may be used for these products; we do not attempt to cover the techniques in this document. We do note that the zPDT **ip1\_dvd** command may be relevant if your Linux for System z distribution is on a DVD and the DVD is configured for direct installation.

A minor restriction exists with SLES distributions prior to SLES 12 Service Pack 1. Earlier distributions will recognize the emulated crypto adapter (in zPDT GA6) as a level CEX4S adapter instead of a level CEX5S adapter.

## 9.3 z/VM

z/VM is available in an AD-CD format for zPDT users whose license includes access to z/VM. For ISV zPDT users (1090 tokens) a separate license agreement may be required. For zD&T users a separately priced feature may be required. Consult your zPDT supplier for additional information.

The zPDT GA8 and GA9 releases corresponds to IBM z14 architecture. Specific service levels of z/VM are required for z14:

- ▶ For z/VM 6.4:
  - The z/VM 6.4 base must be dated on or after August 25, 2017, or
  - APAR VM65942 or APAR VM66071 must be applied to earlier copies of z/VM 6.4.
- ▶ z/VM 6.3 cannot be *installed* on a z14 system. Previously installed z/VM 6.3 systems can be *used* on a z14 if APARS VM65942, VM65921, and VM65922 are installed before attempting to use the system on a z14 machine.
- ▶ Older z/VM releases (before z/VM 6.3) are not considered here.

The current AD-CD z/VM 6.4 system is configured for use with z14 architecture. The details in this chapter are based on this AD-CD z/VM 6.4 system and there may be minor differences with earlier or later releases.

## 9.4 Installing the AD-CD z/VM 6.4 system

The z/VM 6.4 AD-CD uses fewer volumes than earlier AD-CD z/VM releases, due to more effective use of 3390-9 volume space. Six 3390-9 volumes are used for the AD-CD z/VM 6.4 system.<sup>1</sup> These volumes are as follows:

```
M01RES 640RL1 M01P01 M01S01 M01W01 VMCOM1
```

As with other AD-CD System z 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. AD-CD distribution files often have lowercase names (m01res.gz). Our examples expand the files with uppercase names (M01RES), but this is not required. We do it to help distinguish emulated 3390 volumes in the directory.

At the time of writing, the IPL volume is not encrypted and is installed with a simple **gunzip** command. Later z/VM releases might have an encrypted IPL volume similar to that used with current AD-CD z/OS releases.

**Important:** The AD-CD 6.4 z/VM system uses “ZVM640” as the password for all the standard z/VM virtual machines.

<sup>1</sup> The AD-CD z/VM 6.4 might presented with additional paging volumes with volsers such as M01P02, M01P03, and so forth. These (if they exist) are optional. If z/VM finds them present, they will be used as additional paging volumes.

## 9.4.1 zPDT devmap

We created a devmap named devmapvm that defines a minimal z/VM system. The addresses in this example are arbitrary; the disk addresses were chosen to avoid conflicts with the z/OS AD-CD IODF addresses. The devmapvm we used is as follows:

```
[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 10 3270 sessions)

[manager]
name awsckd 0101
device 0200 3390 3990 /z/M01RES
device 0201 3390 3990 /z/VMCOM1      (Address of VMCOM1 is important)
device 0202 3390 3990 /z/640RL1
device 0203 3390 3990 /z/M01S01
device 0204 3390 3990 /z/M01P01
device 0205 3390 3990 /z/M01W01
(Other disks, LAN interfaces, tape drives might also be defined)
```

## 9.4.2 zPDT sensitivity

z/VM 6.4 uses advanced channel commands for its paging functions. zPDT releases prior to GA8 do not support these channel commands. A “standard” z/VM 6.4 system will not work on zPDT releases prior to GA8. However, a special parameter is available in z/VM 6.4 that causes it to use older channel commands for paging. With this parameter, z/VM 6.4 should work correctly with earlier zPDT releases.

This parameter is “PAGING63” and is placed in the IPL parameters line in the Stand Alone Program Loader screen. This has been done in the AD-CD z/VM 6.4 release. You should be aware that a z/VM 6.4 system obtained elsewhere might not have this parameter set and consequently would not work with zPDT releases older than GA8. (You could, of course, insert the parameter in the Program Loader screen to overcome the problem.)

## 9.5 IPL and logon

We used the following zPDT command to IPL this system:

```
$ ip1 200 parm 0700
```

The 0700 is the address of a 3270 terminal. The initial IPL may produce the stand-alone loader panel shown in Figure 9-1 or it may go directly to the OPERATOR session, depending on z/VM customization. If the stand-alone loader panel is displayed, the IPL parameters (shown in Figure 9-1) may need to be changed. The pdvol parameter must point to the address of the VMCOM1 volume. If you want the OPERATOR display at an address other than 700 you can add “cons=xxxx” to the IPL parameters.

```

STAND ALONE PROGRAM LOADER: z/VM VERSION 6 RELEASE 4.0
DEVICE NUMBER: 0200   MINIDISK OFFSET: 39   EXTENT: 1
MODULE NAME:  CPLOAD   LOAD ORIGIN: 1000
-----IPL PARAMETERS-----
pdvol=201 pdnum=1 PAGING63
-----COMMENTS-----

9= FILELIST 10= LOAD 11= TOGGLE EXTENT/OFFSET

```

Figure 9-1 Stand Alone loader

After changing any Stand Alone loader IPL parameters, press PF10 to load z/VM. The local 3270 you specified (0700) is automatically logged on as OPERATOR. The initial panel should look something like Figure 9-2.

The default Stand Alone loader IPL parameters can be changed by logging on as MAINT620 and entering a command similar to the following:

```
SALIPL 123 (EXTENT 1 IPLPARMS pdnum=1 pdvol=0201 cons=0700
```

This command changed the default IPL parameters (in the Stand-Alone Loader) to the values indicated. User MAINT cannot be logged on when using this command.

During z/VM startup you can “warm start” or “cold start” the system. We suggest you always use warm start unless you have a very specific reason for cold starting z/VM. Do not confuse z/VM cold start with the z/OS AD-CD cold start functions. You might need to reply WARM at an operator prompt and N0 for a prompt about the time-of-day clock; typically, pressing Enter is all that is needed for these prompts.

```

10:13:24 z/VM V6 R4.0 SERVICE LEVEL 1601 (64-BIT)
10:13:24 SYSTEM NUCLEUS CREATED ON 2016-10-06 AT 23:55:42, 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, 2016. 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 *****
10:13:24 * IBM z/VM Single System Image Feature is enabled and active.
10:13:24 *****
10:13:24
10:13:24 HCPZC06718I Using parm disk 1 on volume VMCOM1 (device 0207).
10:13:24 HCPZC06718I Parm disk resides on cylinders 1 through 120.
10:13:24 Start (Warm|Force|CoOLDCLEAN) (DRain) (DIsable) (NODIReect)
          (NOAUTOlog) or (SHUTDOWN)

```

CPREAD ZVM640

Figure 9-2 Initial OPERATOR display

The CPREAD at the bottom of a screen means that z/VM is waiting for your input; during startup you can usually simply hit the 3270 Enter key. “MORE . . .” or “HOLDING” at the bottom of a panel indicates that you should clear the screen.<sup>2</sup> You may need to clear it several times until all the initial OPERATOR messages are displayed. The “ZVM640” at the bottom right of the screen is the z/VM system name. AD-CD systems sometimes use this name as the default password,<sup>3</sup> although your AD-CD z/VM system might be different.

At this point you should have the z/VM logo displayed on any other active 3270 sessions, as shown in Figure 9-3. If the logo display is not present, try entering an **ENABLE ALL** command in the OPERATOR session.

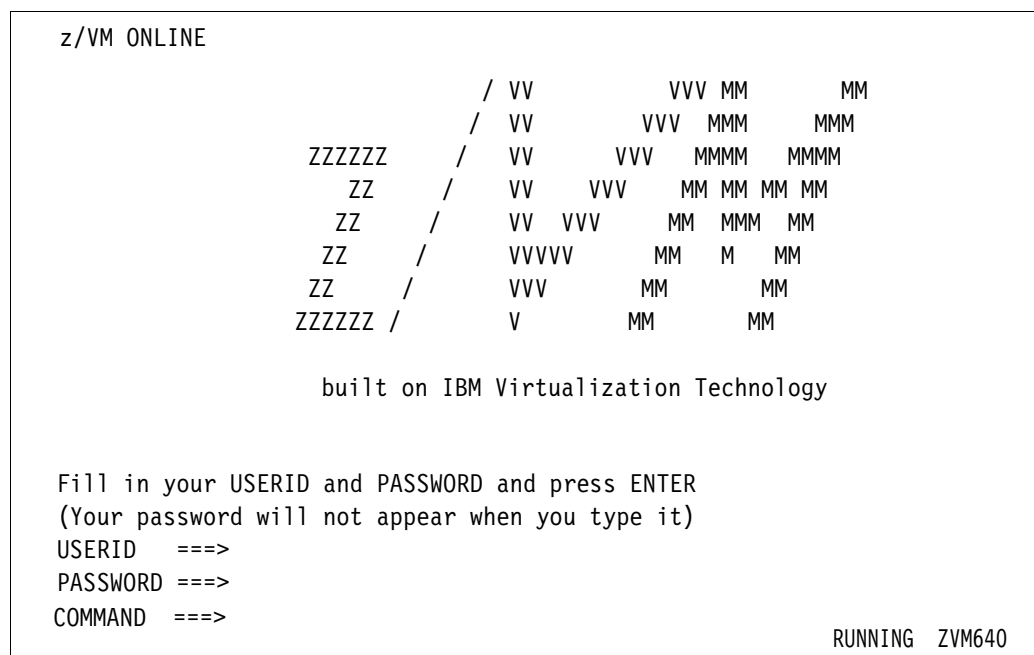


Figure 9-3 z/VM logo display

The z/VM logo panel is commonly used in two ways. The most basic way is to enter a z/VM userid and password. An alternative use is to enter **DIAL xxxx** in the command line, where xxxx is the name of a running z/VM guest that accepts 3270 connections. No userid or password is needed when using a **DIAL** command.

You can disconnect from the OPERATOR session by entering a **DISC** command. This 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 userid is OPERATOR.

During the initial screens during 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 take any actions for them.

## Shutdown

To stop z/VM issue the **SHUTDOWN** command in the OPERATOR session or in another session that has sufficient privileges.

<sup>2</sup> zPDT users of x3270 often set the PAUSE or END key (on their PC keyboard) to perform a 3270 clear operation. If this is not done, then Alt+C may perform the clear function with some emulator setups.

<sup>3</sup> This is obviously terrible security; you can change passwords by editing the VM directory or using DIRMAINT.

## 9.6 Quick z/VM review

**Important:** The remainder of this z/VM material is for almost-new z/VM users. We make no attempt to provide a z/VM primer or to cover many details. The following material can be regarded as “reminders” for some of the more basic details.

### 9.6.1 CMS

CMS is the Conversational Monitor System and is a small, single-user operating system. Every user running CMS is running a separate CMS in a separate virtual machine. (In practice, there is a single copy of CMS in shared virtual memory.) CMS is a special-case operating system that runs only under z/VM. z/VM administration is done through CMS.

Basic CMS use 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 use is not described here.

After IPLing z/VM and logging on with a userid, you need to IPL CMS within that userid (assuming you want to use CMS). This CMS IPL can be automatic (when you log on to the userid) or it can be done manually with an **IPL CMS** or **IPL 190** command. The automatic IPL is enabled by a line in the z/VM directory entry for the userid. A **RUNNING** indicator in the lower right corner of the 3270 panel is an indicator that CMS is running.

CMS has a large set of functions and commands and the basic z/VM Control Program (CP) also has a large set of commands. In general, both sets of commands can be entered on the CMS command line. In a few cases, CP commands must be prefixed by the letters **CP** or **#CP** followed by a space and then the command.

When you are working with z/VM, your virtual machine is in one of three environments:

- ▶ You (or your logon profile) IPLed CMS and you may enter CMS and CP commands.
- ▶ You IPLed another operating system, such as z/OS, and is under the control of that operating system.
- ▶ You did not IPL anything. Only CP commands may be entered. (At this point you might want to IPL CMS in your virtual machine.)

When using CMS (or interacting directly with CP, if CMS has not been started) the 3270 session is in a pseudo-3215 mode. This is a typewriter-like interface, with a command line at the bottom of the panel. Some CMS functions, such as **XEDIT**, provide a full-panel interface similar to **ISPF**.

### 9.6.2 User MAINT

z/VM has a predefined userid, **MAINT**, and most basic z/VM administration is performed while using this userid and working through CMS.<sup>4</sup> In a large production operation, **MAINT** would be used sparingly.<sup>5</sup> In our small sandbox z/VM systems, we use **MAINT** frequently. Among other things, **MAINT** can edit the z/VM directory and activate a new copy. The directory is

<sup>4</sup> Some administration is also done through userid **MAINT620**.

<sup>5</sup> **MAINT** is quite similar to **IBMUUSER** in an initial z/OS installation and is somewhat like *root* in a Linux system. These user IDs have all the necessary authority to further customize and manage the systems. In production operations, various authorities are delegated to other user IDs, and **MAINT** (z/VM) and **IBMUUSER** (z/OS) are seldom used.

where z/VM users are defined, along with details of each user’s virtual machine. An example of directory updating is presented later.

Note that the terms *z/VM users*, *guests*, *userid*s, and *virtual machines* are used interchangeably.

### 9.6.3 Minidisks and files

z/VM works with three types of disk volumes (meaning “real” 3390 volumes):

- ▶ CP-owned volumes. These contain the minidisks needed to run z/VM, or contain paging space, spool space, or temporary space. They may also contain user minidisks.
- ▶ User-owned volumes. These typically contain minidisks but not those used by the operating system.
- ▶ Other disks. These can be used as standard (non z/VM) volumes or as whole-pack minidisks. The z/OS volumes we sometimes use under z/VM are normally defined as whole-pack minidisks. The whole-volume minidisk concept is needed for sharing volumes among multiple z/VM virtual machines, such as multiple z/OS systems.

A minidisk is a contiguous range of 3390 cylinders<sup>6</sup> that is seen by a z/VM user as a complete disk volume (typically with a small number of cylinders). There are three common formats for a minidisk:

- ▶ CMS, which includes all the z/VM control files and user files
- ▶ z/OS, which includes a label, VTOC, and the usual z/OS types of data sets
- ▶ Linux for System z

A minidisk is created by defining it in the z/VM directory. You are responsible for formatting your own minidisks. For CMS this is done with a **format 191 a** command, for example. For z/OS, the ICKDSF utility is used.

A minidisk has a virtual address, such as 190, that is assigned in the z/VM directory. Some minidisk addresses have predefined or conventional meanings. For example, address 190 contains many CMS modules (and is read-only for most users). Most CMS users have a 191 minidisk, which is their default CMS work disk. There are other well-known minidisk addresses that are used by multiple users. Otherwise, a minidisk address is simply a hexadecimal number (usually three digits) that you select.

To access a minidisk under CMS, an access mode letter must be assigned. For example, if you have minidisk 456 defined in your directory entry, you might enter the CMS command **acc 456 z** to access the minidisk 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 minidisks are searched in order, 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 you log on and access your 456 minidisk as your “r” disk.

A CMS user identifies a file with three qualifiers: a file name, a file type, and a mode (which is usually just the drive letter for the minidisk containing the file). The name is not case sensitive.

```

MYDATA  TEXT  A1
|        |    |
|        |    | +---File mode (fm) (The notation fn, ft, fm is common)
|        |    | +-----File type (ft)
+-----File name (fn)
```

<sup>6</sup> Geometry other than 3390 can be used, but 3390s are the most common.

The first part of the name (MYDATA) is completely arbitrary and should be meaningful to you. The second part (TEXT) *can* be arbitrary, but some applications attach meaning to this part of a file name. For example, if the second part of the name is EXEC, it is assumed to contain commands for a command interpreter such as REXX. The third part (A1) has a disk letter (A) and might have a file mode number (1 in the example here) that can be:

- ▶ 0: This makes the file private. Other users with read-only access to your minidisk will not see this file. Users with read-write access will see it.
- ▶ 1: This is the default file mode number for read-write files.
- ▶ 2 and 5: These function like number 1 and can be used to create a subset.
- ▶ 3: Files with this number are automatically erased when they are read.
- ▶ 4: These files are in a simulated MVS format (not used in any of our examples).
- ▶ 6: This indicates the file should be updated in place, with appropriate programming.

For practical purposes, you can omit the mode number and let it default to 1. This is suitable for almost all purposes.

## 9.6.4 Inspecting your disks

Assuming you are logged onto z/VM as MAINT or as another userid, you can use **q da all** and **q disk** commands as shown in Figure 9-4.

```

q da all
DASD 0200 CP OWNED  M01RES  90
DASD 0201 CP SYSTEM 64ORL1  12
DASD 0203 CP SYSTEM M01W01  11
DASD 0204 CP OWNED  M01S01   1
DASD 0205 CP OWNED  M01P01   0
DASD 0206 CP OWNED  VMCOM1  13
DASD 0A80 HDRES1 , DASD 0A81 HDRES2 , DASD 0A82 HDSYS1 , DASD 0A83 HDUSS1
An offline DASD was not found.
Ready; T=0.01/0.01 10:19:15

q disk
LABEL  VDEV M  STAT  CYL TYPE BLKSZ  FILES  BLKS USED-(%) BLKS LEFT  BLK TOTAL
MNT191 191  A   R/W   175 3390 4096    3      10-01    31490    31500
MNT5E5 5E5  B   R/O   18 3390 4096   136    1378-43    1862     3240
MNT2CC 2CC  C   R/W   10 3390 4096    3     110-06    1690     1800
etc...

Ready; T=0.01/0.01 10:19:19

```

*These are online disks that are  
not owned by VM. They might be  
for a z/OS system, for example.*

RUNNING    ZVM640

Figure 9-4 Inspecting the disks

The **q dasd all** command displays all the physical DASD volumes connected to z/VM. If they are in a z/VM format, more information is displayed. In the example here, volumes at addresses 200-206 are in formats recognized by z/VM. Volumes at addresses A80-A9F have standard labels but are not in z/VM formats.

The **q disk** command displays this user's minidisks. In this example, the user has access to the minidisks listed. A minidisk label (MNT191 for the first minidisk) is not important, but might be meaningful to the user. The VDEV value is the "address" of the minidisk. The mode letters (A, B, C, D, E, S, S=Y/S) were probably established by the user PROFILE file that is automatically executed when the user logs onto the system. The meanings of the other fields are fairly obvious.

The CP-owned disks contain a mixture of paging space, temporary disk space (TDISK), spool space, directory space (DRCT), minidisks, and unused space. The **q alloc all** and **q alloc map** commands can display this information.

You can list the files on a minidisk with a command such as **filelist \* \* a**. This example says to list all file names (first asterisk) and all file types (second asterisk) on the minidisk currently accessed as drive a. An example is shown in Figure 9-5.

MAINT FILELIST A0 V 169 Trunc=169 Size=5 Line=1 Col=1 Alt=0									
Cmd	Filename	Filetype	Fm	Format	Lrec1	Records	Blocks	Date	Time
x	USER	DISKMAP	A1	F	100	360	9	2/26/13	9:13:57
	SETUP	\$LINKS	A1	V	26	33	1	9/27/12	17:09:13
	PROFILE	EXEC	A1	V	72	32	1	9/23/11	12:38:14
	SYN	SYNONYM	A1	F	80	1	1	1/15/03	9:46:33
	PROFILE	XEDIT	A1	V	45	17	1	11/18/98	12:26:20
1= Help 2= Refresh 3= Quit 4= Sort(type) 5= Sort(date) 6= Sort(size)									
7= Backward 8= Forward 9= FL /n 10= 11= XEDIT/LIST 12= Cursor									
=====								X E D I T 1 File	

Figure 9-5 FILELIST command

The output from **filelist** is in a full-panel format that can be scrolled with PF8 and PF7. PF3 is used to exit from the command. This format is very useful because CMS commands can be entered in the first column of a line and the file named in that line becomes the operand of the CMS command. Two common commands are **browse** and **x** (or **xedit**).

When a new CMS disk exists (perhaps with a new z/VM userid), it must be formatted for CMS use. Assuming the new disk is my 191 disk (the default “A” disk for a CMS user), at my first logon I could use this dialog:

```
ipl CMS                (if not automatically IPLed at logon)
format 191 a
Format will erase all files on disk A (191). Do you wish to continue?
Enter 1 (YES) or 0 (NO).
1
Enter disk label:
B11191                (the label is arbitrary; 6 characters)
F0rmatting disk A
10 cylinders formatted on A (191)
```

## Another user’s minidisk

You might want to access another user’s minidisk, assuming it is not password-protected. If user Joe has minidisk 456 defined, you might do the following tasks:

```
link joe 456 456      (link to joe’s 456 as my own 456)
acc 456 j              (access it as my j disk)
filelist * * j        (see what files are on the minidisk)
```

There are two minor problems with this method. What if you already have a minidisk defined at address 456?<sup>7</sup> What if you are already using file mode j? Another method is this:

```
vm1ink joe 456        (use the vm1ink command instead)
```

<sup>7</sup> You may use a different address, such as LINK JOE 456 789, but you then must determine whether you already have a 789 disk. This is a trivial problem for most users, who have very few minidisks defined, but it might be a significant problem for some users.

```
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 minidisk you can:

```
rel z                (free file mode z)
det 120              (detach the minidisk from my session)
```

## 9.6.5 XEDIT

XEDIT is the normal CMS editor. Most z/VM administration and customization is done by editing CMS files. New z/VM users who are familiar with ISPF will find XEDIT easy to use once a few differences are mastered. You can create new CMS files with XEDIT simply by naming it. For example, **x newfile stuff a** creates a new file named newfile stuff and place it on your "a" disk (assuming you save the file before exiting from XEDIT).

Important XEDIT commands (on the command line) are as follows:

```
file: Save the current file
qquit: Exit without saving the current file
/xxx/: locate the next line containing characters xxx
+nn: Scroll forward nn lines
-nn: Scroll backward nn lines
top: Scroll to beginning of the file
bottom: Scroll to the end of the file
nulls on or nulls off: Select 3270 nulls/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); exit from Help function
- ▶ PF4: Tab 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

Several XEDIT line commands (overtyping the ===== field in the window) are as follows:

<b>d</b> or <b>dnn</b>	Delete one or <i>nn</i> lines.
<b>i</b> or <b>inn</b>	Insert one or <i>nn</i> lines.
<b>"</b> or <b>"nn</b>	Repeat the line one or <i>nn</i> times.
<b>dd</b> followed by <b>dd</b> in a later line	Delete the indicated block of lines.
<b>cc</b> followed by <b>cc</b> in a later line	Copy the indicated block. The target is noted with <b>p</b> (prior to this line) or <b>f</b> (following this line).

XEDIT has many more commands and facilities than mentioned here, but these few commands might be sufficient for initial use.

## 9.6.6 z/VM directory

z/VM users and minidisks are defined in the z/VM directory. There are two forms of the directory: the *source file*<sup>8</sup> and the *active directory*. A special command reads the source file and creates (or updates) the active directory. Changing the source directory has no effect until the command is executed to create a new active directory.

If you log on as MAINT, you can access the source directory with one of these commands:<sup>9</sup>

```
browse user direct c      (browse it)
x user direct c           (edit it))
```

z/VM offers the DIRMAINT tool that is typically used to maintain the z/VM directory. For very small z/VM systems, such as a sandbox zPDT system, we can “manually” work with the directory.<sup>10</sup> We suggest you browse the directory on your z/VM system to obtain a general look at it. A quick look includes the following items:

- ▶ The PROFILE stanzas define lines that may be included in user definitions by using an INCLUDE statement.
- ▶ Skip over sections such as USER \$DIRECT\$ NOLOG. These help produce a clean disk map.
- ▶ The first real user definition might be for MAINT. The first line of this section begins with the keyword IDENTITY. This is a new keyword that is significant for multiple linked z/VM systems (new with release 6.2). 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 minidisk owned by another user. For example, in MAINT’s directory definitions:

```
LINK PMAINT 2CC 2CC MR
```

means that the minidisk at address 2CC in user PMAINT’s definitions is used at address 2CC in MAINT’s virtual machine.

A simple user definition might be added as shown in Figure 9-6 (these lines would be added to the directory between two existing user entries, or at the end).

---

<sup>8</sup> There could be multiple source files, but we ignore this detail here.

<sup>9</sup> By default, the source file is named USER DIRECT and the logon profile for MAINT accesses the volume as c.

<sup>10</sup> 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 *
SPOOL 00E 1403 A
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 6119 10 VMCOM1 MR
*

```

*Do not copy this statement !*

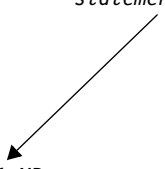


Figure 9-6 Simple user definition

In this example, the userid is BILL and the password is W2WO. The virtual machine is 128 MB, which is more than ample for CMS. The IPL statement causes an automatic IPL of CMS when user BILL logs onto 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 minidisks of other users (all in read-only mode). The specific LINKs shown here provide basic CMS functionality.

The MDISK statement defines a new minidisk at BILL's address 191. (This is the traditional address for a user's A disk.) This statement specifies that the minidisk is placed on a 3390 with volser VMCOM1, with the minidisk starting on cylinder 6119 and it is 10 cylinders long. The MR operand specifies basic read/write access.

How was the cylinder number (6119) obtained? A command is used to map all the minidisks defined in the directory; new minidisks then can be defined on free cylinders. (On a larger, production z/VM system all this is usually done by the DIRMAINT program which handles cylinder addresses automatically.) The following command (run by MAINT) runs the **diskmap** program against the file `user direct c`, which is the default directory source file:

```
diskmap user direct c
```

The output of the program is the file `user diskmap a`. This may be browsed, as follows:

```
browse user diskmap a
```

Inspecting this output, we found that volume VMCOM1 had no minidisks defined after cylinder 6118.<sup>11</sup> We added the MDISK statement for user BILL and ran the **mapdisk** program again, to verify that the new minidisk was correctly mapped. After updating the `user direct c` file, you can activate this new directory; this builds a new *working directory* for z/VM. The command is as follows:

```
directxa user direct c
```

Changes to the directory are not effective until activated by **directxa**. Be certain to look for any error messages.<sup>12</sup>

<sup>11</sup> This was true when this description was written. Subsequent z/VM releases have undoubtedly changed the locations and quantity of free space in the system disks.

<sup>12</sup> When the user logs on and attempts to use the new minidisk it will indicate an error. The new minidisk must be formatted before it can be used. This is done with the **format** command.

Attempting to manage absolute cylinder addresses and ranges when defining minidisks can appear messy and crude. It is. This is why higher-level tools such as DIRMAINT exist; however, using these tools requires additional skills. A very small z/VM, with only a few simple added users, can be readily managed by directly editing the z/VM directory. A few guidelines include:

- ▶ Do not change the minidisk definitions (or paging, spooling, temporary disk space, or directory space) for the system volumes, with the exception that you can add a few small minidisks in empty space on volumes such as M01W01.
- ▶ Use the diskmap program frequently and look for *overlap* or *gap* flags. Incorrect cylinder specifications can overlap two minidisks, usually resulting in corruption of both.
- ▶ Never allocate a minidisk on cylinder zero. (This restriction does not apply to full-volume minidisks.)

## 9.6.7 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. In general, simple use of z/VM as a base for running multiple z/OS guests will not involve these devices. However, various internal z/VM services (such as TCP/IP) write logs to the virtual printer or send messages and logs as input to the virtual card reader. You may want to review this information. In the fullness of time (probably a long time), these could fill the spool space provided with the distributed AD-CD z/VM system.

One way to view the contents of the virtual card reader and printer queue is with the commands:<sup>13</sup>

```
q rdr
q prt
q pun          (although there are seldom any virtual punch files)
```

with output similar to that shown in Figure 9-7.

Ready; T=0.03/0.08 10:27:04										
<b>q rdr</b>										
OWNERID	FILE	CLASS	RECORDS	CPY	HOLD	USERFORM	OPERFORM	DEST	KEEP	MSG
MAINT	0027	T	CON	00000022	001	NONE	STANDARD	STANDARD	OFF	OFF
PMAINT	0005	T	CON	00000075	001	NONE	STANDARD	STANDARD	OFF	OFF
...										
MAINT	0031	T	CON	00000053	001	NONE	STANDARD	STANDARD	OFF	OFF
OPERATNS	0001	D	SYS	00000000	001	NONE			OFF	OFF
OPERATNS	0002	D	SYS	00000000	001	NONE			OFF	OFF
Ready; T=0.01/0.01 10:27:08										
<b>q prt</b>										
OWNERID	FILE	CLASS	RECORDS	CPY	HOLD	USERFORM	OPERFORM	DEST	KEEP	MSG
OPERATOR	0025	T	CON	00000077	001	NONE	STANDARD	STANDARD	OFF	OFF
DTCVSW1	0018	T	CON	00000038	001	NONE	STANDARD	STANDARD	OFF	OFF
DTCVSW2	0018	T	CON	00000038	001	NONE	STANDARD	STANDARD	OFF	OFF
MAINT	0032	T	CON	00000123	001	NONE	STANDARD	STANDARD	OFF	OFF
Ready; T=0.01/0.01 10:27:14										
										RUNNING zVM640

Figure 9-7 Spooled files

The first column indicates the z/VM userid that owns the file.

<sup>13</sup> Remember that each user potentially has his own virtual reader, printer, and punch data.

You can list the spooled rdr files that you own with the **rdrlist** command, as shown in Figure 9-8 on page 186.

MAINT RDRLIST A0 V 164 Trunc=164 Size=13 Line=1 Col=1 Alt=2										
Cmd	Filename	Filetype	Class	User	at	Node	Hold	Records	Date	Time
*	(none)	(none)	CON	T	DTCVSW1	SSI1	NONE	48	2/26	8:49:33
	(none)	(none)	CON	T	DTCVSW2	SSI1	NONE	48	2/26	8:49:33
<b>discard</b>	(none)	(none)	CON	T	MAINT	SSI1	NONE	78	2/26	8:50:17
=	(none)	(none)	CON	T	DTCVSW1	SSI1	NONE	38	2/28	9:22:37
=	(none)	(none)	CON	T	DTCVSW2	SSI1	NONE	38	2/28	9:22:37
=	(none)	(none)	CON	T	MAINT	SSI1	NONE	20	2/28	10:33:32
.....										
	(none)	(none)	CON	T	DTCVSW1	SSI1	NONE	53	3/02	12:29:22
1= Help 2= Refresh 3= Quit 4= Sort(type) 5= Sort(date) 6= Sort(user)										
7= Backward 8= Forward 9= Receive 10= 11= Peek 12= Cursor										
====> X E D I T 1 File										

Figure 9-8 Display from the **rdrlist** command

The **rdrlist** display is quite useful. By moving the cursor to one of the lines and pressing PF11 you can peek (view) the contents of the file. You can discard files by using the **discard** line command; entering equal signs (=) on other lines indicates they are to have the same treatment as the command (the **discard** command in this case). Unfortunately, there is no equivalent **prtlist** 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
```

On a larger scale, the following commands may be used by an authorized user (such as MAINT) to purge many or all spooled files:

<b>purge &lt;userid&gt; rdr &lt;file number&gt;</b>	<i>(purge a specific rdr spool file)</i>
<b>purge &lt;userid&gt; rdr ALL</b>	<i>(purge all the user's rdr files)</i>
<b>purge &lt;userid&gt; prt &lt;file number&gt;</b>	<i>(purge a specific prt spool file)</i>
<b>purge &lt;userid&gt; prt ALL</b>	<i>(purge all the user's prt files)</i>
<b>purge system rdr ALL</b>	<i>(purge all the rdr files in spool)</i>
<b>purge system prt ALL</b>	<i>(purge all the prt files in spool)</i>

The <userid> may 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.

## 9.6.8 Simple system queries

A number of simple query, display, and access commands might be useful:

- ▶ **q disk**: List your minidisks.
- ▶ **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; command limited to certain userids.
- ▶ **q accessed**: List minidisks you have accessed.
- ▶ **q links 120**: List userids who have links to my 120 disk.
- ▶ **q pf**: List Program Function Key assignments.
- ▶ **q stor**: How much System z storage?

- ▶ **q n**: Which userids are logged on?
- ▶ **q all**: What disks and terminals are online?
- ▶ **q rdr**: List the files in your virtual reader.
- ▶ **q prt**: List files in your virtual print queue.
- ▶ **set pf12 retrieve**: Make PF12 function as 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 particular reader file belonging to userid joe.
  - **purge joe rdr all**: Purge all 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 my 456 disk.
- ▶ **acc 456 j**: Access my 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 my userid
- ▶ **vmLink joe 345**: a Combined **link** and **acc** function.
- ▶ **format 191 a**: Format a new minidisk.
- ▶ **directxa user direct c**: Activate an updated z/VM directory.
- ▶ **ind**: How busy is the system?
- ▶ **diskmap user direct c**: Create a minidisk map based in directory user direct c.
- ▶ **browse user diskmap a**: Inspect a file

## 9.6.9 zIIPs and zAAPs

z/VM can provide *simulated* zIIPs or zAAPs, working only with CPs in the base zPDT system. Furthermore, z/VM can provide more logical CPs, zIIPs, and zAAPs than there are CPs present in the base zPDT system.

The definition of a z/VM guest, in the z/VM directory, can contain statements such as these:

```
MACH ESA 5                (allow up to 5 logical processors)
CPU 0 BASE
COMMAND DEFINE CPU 1 TYPE ZIIP
```

The two logical processors (one CP, one zIIP) can be used even if the base zPDT definition has only a single CP (a model1090-L01, for example). Of course, there are performance implications if the number of logical processors greatly exceeds the number of “real” System z processors, but this may be acceptable for development and testing situations.

## 9.6.10 Paging

If you run z/OS (or another System z operating system) under z/VM on a zPDT base machine, remember that you potentially have three levels of paging:

- ▶ The base Linux system pages whenever virtual memory usage exceeds the available real memory. Our advice to have *at least* 1 GB more real memory than your defined for System z memory size. This is intended to minimize Linux paging. A Linux page fault in the primary zPDT CP process causes the CP to pause until the page fault is resolved.
- ▶ z/VM pages when its requirement for virtual memory exceeds the defined System z memory (which is actually base Linux virtual memory). Each z/VM guest (whether a CMS

user or a whole z/OS system) resides in z/VM virtual memory. z/VM systems on larger machines, running multiple significant guests, tend to page rather heavily.

- ▶ z/OS (or another System z operating system) pages when its need for real memory (which is actually z/VM virtual memory<sup>14</sup>) exceeds whatever size was defined in the z/VM directory for the z/OS guest.

A zPDT system has limited I/O bandwidth to its disk, and that bandwidth is best used for running applications rather than for paging. Our advice is to consider your memory usage carefully when planning to use z/VM for multiple guests.

### 9.6.11 Simple network adjustments

The following are brief directions<sup>15</sup> for setting or changing the IP address for z/VM and for enabling FTP functions. We use arbitrary addresses for OSA that are different from those typically used by the AD-CD z/OS system. We selected addresses 0240-0242. We added the following lines to the devmap we used for z/VM:

```
[manager]
name awsosa 1100 -path=A0 --pathtype=OSD --tunnel_intf=yes
device 0240 osa osa
device 0241 osa osa
device 0242 osa osa
```

Logon to MAINT and issue these commands to access key TCPIP files:

```
cp link tcpmaint 198 198 mr
acc 198 w
filelist * * w
```

Edit the SYSTEM DTCPARMS file as follows:

```
:nick.TCPIP      :type.server
                  :class.stack
                  :attach.0240 AS 0001,0241 AS 0002,0242 AS 003
```

Then edit PROFILE TCPIP to include the following changes:

```
PROFILE TCPIP
  PORT 20 TCP FTPSERV          (uncomment this line; remove NOAUTOLOG)
  PORT 21 TCP FTPSERV          (uncomment this line)
  ...
  DEVICE DEV@0240 OSA 0001 NONROUTER
  LINK OSA0240 QDIOETHERNET DEV@0240 PATHMTU MTU 1492 IP
  HOME
  10.1.1.99 255.255.255.0 OSA0240      (use your desired IP address!)
  GATEWAY
  DEFAULTNET 10.1.1.1 OSA0240 1492      (use your gateway IP address)
  START DEV@0240
```

You can logon to TCPIP and IPL CMS to restart TCPIP with the new parameters, but this can become a little complicated and it might be easier to reIPL z/VM.

To use network functions, such as **ping**, **netstat**, or **ftp** you must access their minidisk:

```
cp link tcpmaint 592 592 rr
```

<sup>14</sup> While conceptually true (ignoring V=R and similar environments), the implementation details depend on the level of assists enabled through the SIE instruction.

<sup>15</sup> As usual, there are many different ways to do this; we list just one of these ways.

```
acc 592 p
netstat home
```

You need to do this from any z/VM userid that intends to use TCP commands.

## 9.7 z/TPF

z/TPF is a specialized IBM operating system used for very high transaction-rate, high-availability applications such as airline reservations, hotel reservations, ATM transactions, and so forth. 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 through zD&T as a limited offering. (The following text uses “zPDT” to mean the zPDT component that is part of the zD&T product offerings.)

Serious inquiries about z/TPF should go to [TPFQA@us.ibm.com](mailto:TPFQA@us.ibm.com).

There is no prepackaged z/TPF system similar to the AD-CD packages for z/OS or z/VM. The assumption is that z/TPF customers will either generate a suitable z/TPF or will migrate one of their working z/TPF systems from another platform to zPDT. For the following discussion we copied<sup>16</sup> a “16-mod” system (meaning 16 3390-3 volumes) and used it for basic z/TPF startup and operation.

z/TPF application development is based on Linux for z Systems, as outlined in Figure 9-9.

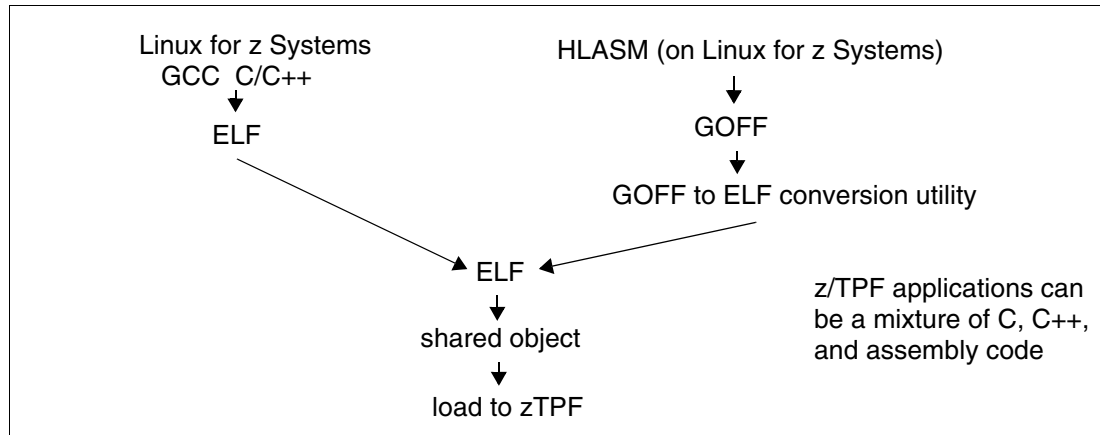


Figure 9-9 z/TPF application development flow

Both Linux for z Systems and z/TPF can be used on a zPDT base. While it is possible to run both these under z/VM in a single zPDT instance, the z/TPF developers recommend using separate zPDT instances. (These separate zPDT instances can be on a single PC or on separate PCs.)

The basic zPDT devmap we used is as follows:

```
[system]
memory 4096m
processors 1
```

<sup>16</sup> We copied the volumes with the zPDT migration utility, working with a z/VM system on the “other end” of the migration link.

```
3270PORT 3270
```

```
[manager]
name aws3215 0
device 0056 3215 3215 #primary command console for z/TPF

[[manager]
name aws3274 4
device 008B 3277 3274 int061 #after start up can be operator console
device 0024 3284 3274 int062 #3284 configuration needed, but not used
```

```
[manager]
name awsosa 12 --path=f8 --pathtype=osd
device 1A0 osa osa
device 1A1 osa osa
device 1A2 osa osa
```

```
[manager]
name awstape 8
device B600 3480 3803 /z/TAPE00 #These are SL tape volumes
device B601 3480 3803 /z/TAPE01 #The tape volumes are empty except
device B602 3480 3803 /z/TAPE02 #for a standard label on each volume
device B603 3480 3803 /z/TAPE03
```

```
[manager]
name awscckd 13
device 1EC0 3390 3990 /z/BJ0001 #IPL volume
device 1EC1 3390 3990 /z/BJ0002
device 1EC2 3390 3990 /z/BJ0003
device 1EC3 3390 3990 /z/BJ0004
device 1EC4 3390 3990 /z/BJ0005
device 1EC5 3390 3990 /z/BJ0006
device 1EC6 3390 3990 /z/BJ0007
device 1EC7 3390 3990 /z/BJ0008
device 1ED0 3390 3990 /z/BJ0017
device 1ED1 3390 3990 /z/BJ0018
device 1ED2 3390 3990 /z/BJ0019
device 1ED3 3390 3990 /z/BJ0020
device 1ED4 3390 3990 /z/BJ0021
device 1ED5 3390 3990 /z/BJ0022
device 1ED6 3390 3990 /z/BJ0023
device 1ED7 3390 3990 /z/BJ0024
```

Operator interaction with z/TPF, especially during initial startup, is somewhat different than what is found in other z Systems operating systems. Operator interaction is through an emulated 3215.<sup>17</sup> Input commands use the zPDT command **awsin** to send text to the emulated 3215, while output text for the emulated 3215 is displayed in the base Linux command window that was used to start zPDT. z/TPF, at times, frequently repeats prompts for various input parameters. This frequent asynchronous output (to the emulated 3215, as displayed in the command window used to start zPDT) makes entering commands in the same Linux command window confusing. We found it easier to use a separate Linux command window to enter z/TPF commands, as illustrated in Figure 9-10.

---

<sup>17</sup> An IBM 3215 was a typewriter-like console terminal.

Every input to the 3215 terminal is with the zPDT **awsin** command. Repeatedly typing this is prone to errors and we used the Linux **alias** command to set + (the plus sign) to mean **awsin**. Thus typing “+ TPF01” is the equivalent of typing “**awsin** TPF01” and takes less time. The arrows in the figure illustrates where 3215 responses to z/TPF prompts is echoed in the 3215 output.

The 3277 defined in the devmap can be configured for z/TPF operator commands after the system is started. The 3284<sup>18</sup> in the devmap is not actually used, but the z/TPF protocol needs to associate it with the 3277. The four tape drives in the devmap (only one was used in our basic z/TPF startup) have premounted tape volumes, as indicated in the devmap. We earlier initialized these four tape volumes with standard labels by using the zPDT **aws\_tapeInit** command.

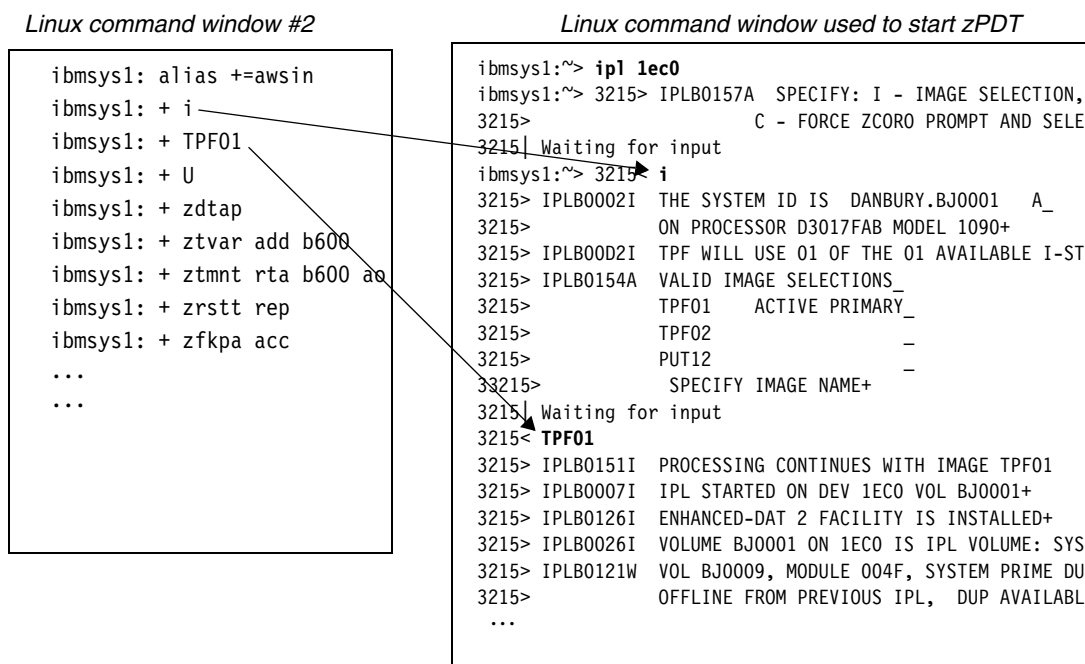


Figure 9-10 Using two Linux command windows for 3215 operation

z/TPF operator commands are rather arcane and we do not attempt to explain z/TPF operation or usage here. Readers familiar with z/TPF might find our basic startup command sequence familiar:

ibmsys1: + <b>alias +=awsin</b>	<to avoid typing “awsin” many times
ibmsys1: + <b>i</b>	<reply to select a z/TPF image
ibmsys1: + <b>TPF01</b>	<we selected an image named TPF01
ibmsys1: + <b>U</b>	<continue the startup
ibmsys1: + <b>zdtap</b>	<display online tape devices
ibmsys1: + <b>ztvar add b600</b>	<vary tape at address B600 online
ibmsys1: + <b>ztmnt rta b600 ao</b>	<mount rta active output on this tape
ibmsys1: + <b>zrslt rep</b>	<continue startup
ibmsys1: + <b>zfkpa acc</b>	<continue startup
ibmsys1: + <b>zdtim</b>	<display hardware TOD clock
ibmsys1: + <b>zatim 1018 tod good</b>	<set z/TPF time value
ibmsys1: + <b>zapat cxml timeout-65535</b>	<set longer watchdog timer for XML
ibmsys1: + <b>zdstat</b>	<display
ibmsys1: + <b>zdsys</b>	<display system status

<sup>18</sup> An IBM 3284 was a small printer that could be connected with a cluster of 3270 terminals.

```

ibmsys1: + zdcrs                                <display I/O device state
ibmsys1: + zacrs rep a17 520000 type-prt cpuid-a    <add 3274 printer
ibmsys1: + zacrs rep a18 4e0000 type-crt cpuid-a prt-A17 <add 3277
ibmsys1: + zdcrs                                <display revised I/O state
ibmsys1: + zcyc1 norm                            <move to normal state
ibmsys1: + zatme good                            <indicate TOD is good
ibmsys1: + zdtcp                                <display tcp status
ibmsys1: + zcyc1 1052                            <move to normal operation state

```

The **zapat cxml timeout-65535** command might not be familiar to experienced z/TPF readers. z/TFP monitors the CPU time used for commands and transactions and terminates functions that take longer than expected. z/TPF startup involves processing a large amount of XML and this takes longer than z/TPF expects when running on zPDT. The **zapat** command extends the timeout period for this XML processing.<sup>19</sup>

The following listing (with many lines omitted for brevity) illustrates that z/TPF startup follows the same pattern under zPDT as it would on a larger z System. The 3215 “prompts” indicate that the output message is displayed on the emulated 3215.

```

ibmsys1: ipl 1ec0
ibmsys1: 3215> IPLB0157A SPECIFY: I - IMAGE SELECTION, B - BYPASS SELECTION_
3215> C - FORCE ZCORO PROMPT AND SELECT IMAGE+
3215| Waiting for input
ibmsys1: 3215< i
3215> IPLB0002I THE SYSTEM ID IS DANBURY.BJ0001 A_
3215> ON PROCESSOR D3017FAB MODEL 1090+
3215> IPLB0002I TPF WILL USE 01 OF THE 01 AVAILABLE I-STREAM +
3215> IPLB0154A VALID IMAGE SELECTIONS_
3215> TPF01 ACTIVE PRIMARY_
3215> TPF02 _
3215> PUT12 _
33215> SPECIFY IMAGE NAME+
3215| Waiting for input
3215< TPF01
3215> IPLB0151I PROCESSING CONTINUES WITH IMAGE TPF01 +
3215> IPLB0007I IPL STARTED ON DEV 1ECO VOL BJ0001+
3215> IPLB0126I ENHANCED-DAT 2 FACILITY IS INSTALLED+
      (additional messages omitted)
3215> IPLB0130I BSS FARF ADDRESSING STAGE IS 3/4, DISPENSE MODE IS 3+
3215> IPLB0031I IPL PROGRAM CALLING CTIN+
3215> CTIN0012I - CT00 INIT STARTED+
3215> CTIN0014I - CT01 TAPE TABLES ALLOCATED AND TSTB INIT+
      (additional normal startup messages omitted)
COTB0176I 10.54.00 TRST BSS TAPE RESTART STARTED +
3215> CSMP0097I 10.54.00 CPU-A SS-BSS SSU-BSS IS-01
COSA0146A 10.54.00 TRST BSS MOUNT RTA TAPE +
3215< zdtap
3215> CSMP0097I 10.54.00 CPU-A SS-BSS SSU-BSS IS-01
COTD0002I 10.54.00 DTAP - TAPE STATUS
      TAPE RESTART INCOMPLETE
ADDRESS NAME SSU STATUS TPIND VOLSER FORMAT #BLOCKS LDR QUEUE
B600 AVAIL
3215> END OF DISPLAY+
3215< ztmnt rta b600 ao
3215> CSMP0097I 10.54.00 CPU-A SS-BSS SSU-BSS IS-01
COTM0310I 10.54.00 TMNT BSS TAPE RTA MOUNTED ON DEVICE B600
      VSN TAPE00 G0358 S0001 F38K SL BZOS 1 NOCOMP NOENC +
3215> CSMP0097I 10.54.00 CPU-A SS-BSS SSU-BSS IS-01

```

<sup>19</sup> The latest z/TPF code might eliminate the need for this timing change.

```

C          (many messages omitted here)
CVRN0004I 10.54.01 RESTART COMPLETED- 1052 STATE+
3215< zdsys
3215> CSMP0097I 10.54.01 CPU-A SS-BSS SSU-BSS IS-01
DSYS0001I 10.54.01 THE SYSTEM IS IN 1052 STATE FOR SUBSYSTEM BSS
          ON BJ0001 CPU-A 26JUL

END OF DISPLAY+
33215< zcycl norm
3215> CSMP0097I 10.54.01 CPU-A SS-BSS SSU-BSS IS-01
CYCL0001I 10.54.01 CYCL TO NORM - STARTED+
ENTER ZATME GOOD TO CONTINUE CYCLE UP OR
ENTER ZATME CNCL TO CANCEL CYCLE UP+
3215< zatme good
3215> CSMP0097I 10.54.01 CPU-A SS-BSS SSU-BSS IS-01
ATME0001I 00.00.00 ZCYCL WILL CONTINUE AS REQUESTED+
          (more messages omitted, including TCP and HTTPPS startup)
INET0050I 10.33.01 MATIPB IS NOW ACCEPTING CONNECTIONS ON
          IP - ANY PORT - 00351 PID - 400102F7+

3215< zcycl norm
3215> CSMP0097I 10.33.56 CPU-A SS-BSS SSU-BSS IS-01
CYCL0005T 10.33.56 INVALID CYCLE REQUEST+
3215< zstat
3215> CSMP0097I 10.34.06 CPU-A SS-BSS SSU-BSS IS-01 _
3215< zcycl 1052
          (remaining messages omitted)

```

z/TPF is a complex package and the material presented here is simply to demonstrate that it can be used under z/TPF. As with any complex package, there is a learning curve involved and users must develop their own techniques of operation.

z/TPF installation is also complex and involves assembling a large set of macros that create the steps to install z/TPF. The following is a *small* excerpt from the assembly job, providing a flavor of the details involved.

```

TITLE 'z/TPF BASE-ONLY SYSTEM '
PRINT NOGEN
*****
*          CONFIG - SYSTEM HARDWARE SUPPORT          *
*****
CONFIG    APRNT=OOE,          ADDRESS OF PRINTER FOR IPL SEQUENCE X
          DUMPDEV=PRTR,      UPON IPL DUMPS ARE SENT TO PRINTER X
          FQTK=NO,          NO FARE QUOTE TICKETING X
          MAPSP=NO,        DO NOT INCLUDE 3270 MAPPING SUPPORT X
          MSGSW=NO,        NO MESSAGE SWITCHING SUPPORT X
          USER=USA,        USER OF PARS X
          DCUSV=64,        ADDRESS RANGE DASD CONTROL RANGE X
          ENTERPRISE=DANBURY, X
          COMPLEX=TPFANET, X
          SYSID=(A),        SDPS X
          VM=YES,          TPF WILL RUN UNDER VM/370 X
          RES=NO,          DO NOT INCLUDE RES APPLICATION X
          TEST=NO,        WP TEST SYSTEM PARMETER X
          ACF=YES,        ACF FEATURE X
          MPIF=YES,        MPIF FEATURE X
          NEF=YES,        ALCI SUPPORT X
          TPFAR=YES,        TPFAR FEATURE X
          FUNCEXT=YES,      FUNCTION RETURN X
          WTOPCUNS=YES,      X

```

```

SELECT=YES,          ENABLE E-TYPE LOADER SEL. ACTIVATE X
APACHE=YES,          INCLUDE APACHE CODE                X
APACHEV2=YES,        INCLUDE APACHE CODE                X
TAR=YES,             INCLUDE TAR/PAX SUPPORT            X
LIBCURL=YES,         INCLUDE LIBCURL LIBRARY            X
MYSQL=YES,           Include MySQL support              X
BPCRLOAD=YES,        Bypass CRPA load in restart        X
ZLIB=YES,            Enable ZLIB support for Apache      X
OPENLDAP=YES,        Enable OpenLDAP support            X
BDB=YES,             Enable BDB support                X
PRRS=NO,             Enable PRRS support                X
CTKI32LC=NO          GENERATE CTKI IN PRE-32LC FORMAT

*****
*                      ***** DASD *****                      *
*****
      IODEV      IOADR=0260,DVTYP=DASD          VPARS
      IODEV      IOADR=03E0,DVTYP=DASD          VPARS
      IODEV      IOADR=0460,DVTYP=DASD          VPARS
      (many lines omitted)
*****
*                      ***** TAPE *****                      *
*****
      IODEV      IOADR=32,DVTYP=TAPE          STAND ALONE 3590
      IODEV      IOADR=42,DVTYP=TAPE          VTAPE
      IODEV      IOADR=43,DVTYP=TAPE          VTAPE
      (many lines omitted)
*****
*                      CORREQ - TPF CORE DEFINITION                      *
*****
CORREQ  MMES=20,          max no of 1M frames/ecb in 31bit X
        XMMES=2,          max no of 1M frames/ecb in 64bit X
        EMPS=20,          max size of 31bit ECB heap      X
        EPRIV=4,          ECB private area size in MB     X
        TRENTY=200,        max no of entries in buffer    X
        ESPS=2,            stack size in MB               X
        AP64CW=200,        size of copy-on-write 64-bit CRPA X
        AP31=80,           size of standard 31-bit CRPA (MBs) X
        AP31CW=80,         size of copy-on-write 31-bit CRPA X
        AP64=80,           size of standard 64-bit CRPA (MBs) X
        MCMTB=1024
CORREQ  MEMCONFIG=IBMSMALL, DEFINE BSS MEMORY CONFIG      X
        FRM=1000,          NUMBER OF 4K FRAMES            X
        CMB=100,           NUMBER OF 4K COMMON BLOCKS     X
        ECBS=50,           NUMBER OF 12K ECBS             X
        IOB=2500,          NUMBER OF I/O BLOCK            X
        SWB=700,           NUMBER OF 4K COMMON BLOCKS     X
        FRM1MB=220,        number of 1 megabyte frames    X
        PEH=1,             preallocated ecb heap          X
        HAVL1=0,           ecb heap available list 1      X
        HAVL2=0,           ecb heap available list 2      X
        HAVL3=0,           ecb heap available list 3      X
        HAVL4=0,           ecb heap available list 4      X
        PPA=1,             preallocated EPA               X
        PES=16,            pre-allocated stack            X

```

	SHP=12,	PRE-ALLOCATED 31BIT SYSTEM HEAP	X
	SHA=0,	PRE-ALLOCATED 64BIT SYSHEAP AREA	X
	VFAMIN=10,	MINIMUM VFA	X
	VFAMAX=0,	MAXIMUM VFA	X
	DBA=1	dump buffer area	
CORREQ	MEMCONFIG=IBMBASMC,	DEFINE BAS MEMORY CONFIGURATION	X
	FRM=7000,	NUMBER OF 40 BYTE I/O BLOCKS	X
	CMB=250,	NUMBER OF 4K COMMON BLOCKS	X
	ECBS=300,	NUMBER OF 12K ECBS	X
	IOB=2700,	NUMBER OF I/O BLOCK	X
	SWB=1250,	NUMBER OF 4K COMMON BLOCKS	X
	FRM1MB=500,	number of 1 megabyte frames	X
	PEH=16,	preallocated ecb heap	X
	HAVL1=64,	ecb heap available list 1	X
	HAVL2=16,	ecb heap available list 2	X
	HAVL3=16,	ecb heap available list 3	X
	HAVL4=2,	ecb heap available list 4	X
	PPA=1,	preallocated EPA	X
	PES=16,	pre-allocated stack	X
	SHP=20,	PRE-ALLOCATED 31BIT SYSTEM HEAP	X
	SHA=0,	PRE-ALLOCATED 64BIT SYSHEAP AREA	X
	VFAMIN=10,	MINIMUM VFA	X
	VFAMAX=0,	MAXIMUM VFA	X
	DBA=1	dump buffer area	
CORREQ	MEMCONFIG=IBM8GIG,	DEFINE BSS MEMORY CONFIG	X
	FRM=9000,	NUMBER OF 4K FRAMES	X
	CMB=500,	NUMBER OF 4K COMMON BLOCKS	X
	ECBS=1200,	NUMBER OF 12K ECBS	X
	IOB=7500,	NUMBER OF I/O BLOCK	X
	SWB=30000,	NUMBER OF 4K COMMON BLOCKS	X
	FRM1MB=1200,	number of 1 megabyte frames	X
	PEH=20,	preallocated ecb heap	X
	....		

(many, many lines omitted)





## Multiple instances and guests

zPDT supports both guest operations (under z/VM) and multiple instances of zPDT. Both are ways to run multiple z/OS or other operating systems.

In this chapter we present basic information about the use of multiple zPDT instances. Practical operation with multiple instances can become complex. You might need to work with your zPDT provider to clarify usage of more complex configurations.

See additional notes elsewhere about multiple instances using LANs and using cryptographic adapter functions.

### 10.1 Multiple instances or guests

We strongly suggest that you use a single zPDT instance, as described in this document, to become familiar with basic zPDT operation. Also, you cannot exceed the number of zPDT licenses in your token (or tokens, or license server). The total number of zPDT licenses applies whether the zPDT CPs are in a single instance or spread over multiple instances.

Multiple *instances* means running more than one copy of zPDT. Each instance must run under a different Linux userid. This can be accomplished by logins through Telnet (or SSH) or by careful use of **su** commands from different windows on the Linux desktop. Each instance must have its own devmap.

The use of TCP/IP interfaces is an essential part of this discussion. For this reason we combine a discussion of multiple zPDT instances with the use of guests under z/VM in a single instance. Our examples use OSD (QDIO) interfaces for TCP/IP. It is possible to use OSE interfaces (non-QDIO) for TCP/IP; however, in the case of multiple instances using OSE (through a single emulated OSA) you *must* configure the OSA Address Table (OAT) using the OSA/SF utility.

## 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, CMS) acquire a 3270 when a user logs on to the guest or uses a **DIAL** command.

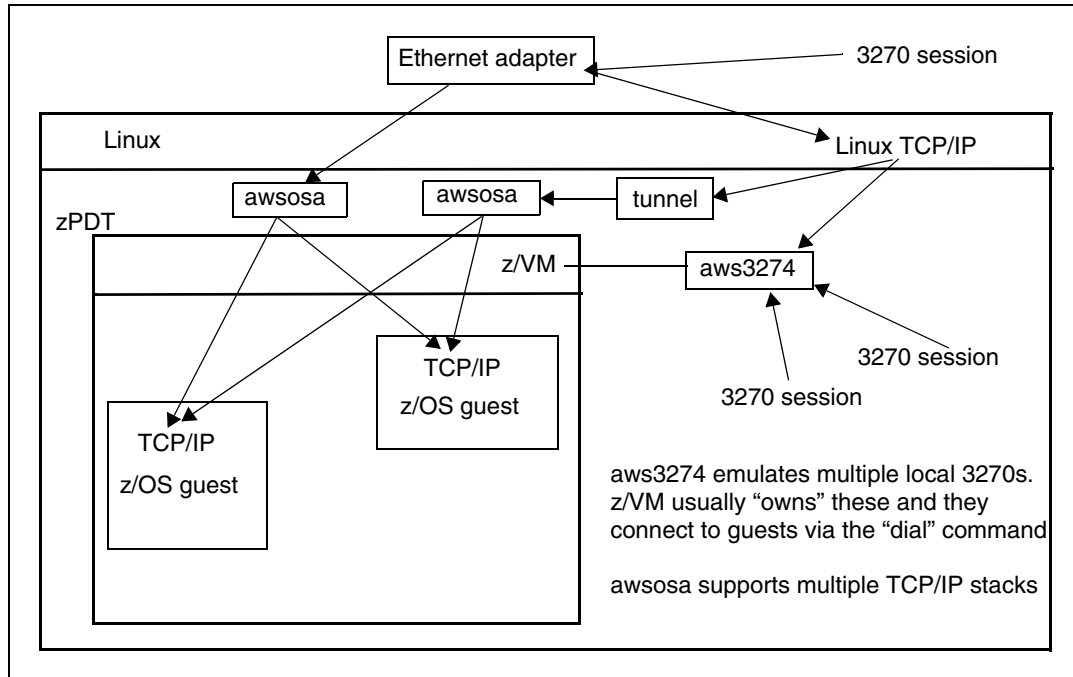


Figure 10-1 Guests in a single zPDT instance

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 using a tunnel to Linux can be used, as shown in the illustration. Each z/VM guest uses different IP addresses on each OSA interface. Alternatively, not shown in the illustration, z/VM could 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:

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 zPDT instances, meaning that emulated I/O devices are not shared between the instances. In common terms, there is no shared DASD (or any other shared device).

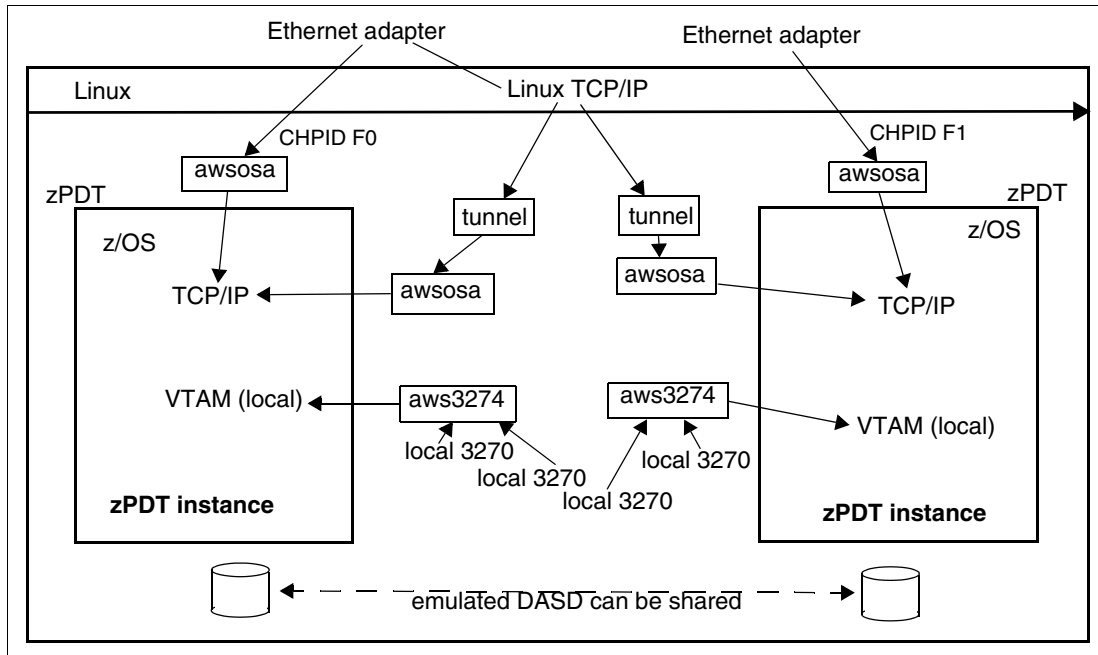


Figure 10-2 Independent instances

In this example, we assume three zPDT licenses (and a base Linux machine with at least four cores) and we have assigned two CPs (a CP and a zIIP) to one instance and one CP to the other instance. Notice that 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, as described here.

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 LCS (non-QDIO) mode for both instances, but they could both be QDIO or a mixture of LCS and QDIO.<sup>1</sup> The following devmaps create a tunnel interface for only one of the instances simply to illustrate that quite different configurations are possible for independent instances.

Simplified devmaps, matching Figure 10-2, might be as follows:

```
(file /home/ibmsys1/aprof1)
[system]
memory 3000m                # emulated zSeries to have 3000 MB memory
3270port 3270                # tn3270e connections specify this port
processors 2 cp ziip          # one CP and one zAAP

[manager]
name awsckd 0001              # define a single 3390 disk
device 0a80 3390 3990 /z/SARES1

[manager]
name aws3274 0003             # define two local 3270s
device 0700 3279 3274 mstcon
device 0701 3279 3274 tso

[manager]
```

<sup>1</sup> Because each instance has its own OSA, the user is not required to do OAT configuration if the user uses the default OAT definitions.

```

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 4GB 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 3270s
device 0700 3279 3274 L700
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

```

Startup for these instances, working from the Linux desktop, might go as follows:

```

(login as root; 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, IPL 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, IPL VM

```

Each instance is started 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 use of **xhost +** presents a security exposure; you should tailor this command to suit your security environment

## 10.4 Instances with shared I/O

A possibility is for multiple instances to share certain devices, such as emulated DASD and emulated OSA. Also, a single pool of 3270 devices can be used and accessed via a common Linux port number, although this option has more complex side effects. The most common use of a shared configuration is to provide *shared DASD* among the instances.

zPDT does not support the VMAC function of z/OS. The only virtual MAC supported is generated on z/VM with the layer-2 vswitch.

A configuration with shared I/O devices requires a *group controller*; see Figure 10-3. The group controller is similar to another zPDT instance, but without an associated CP or memory. The group controller must have its own Linux userid, its own devmap, and be started with its own **awsstart** command. It must be started before other instances are started. As a basic concept, the I/O devices defined in the group controller's devmap are inherited and shared by the other instances.

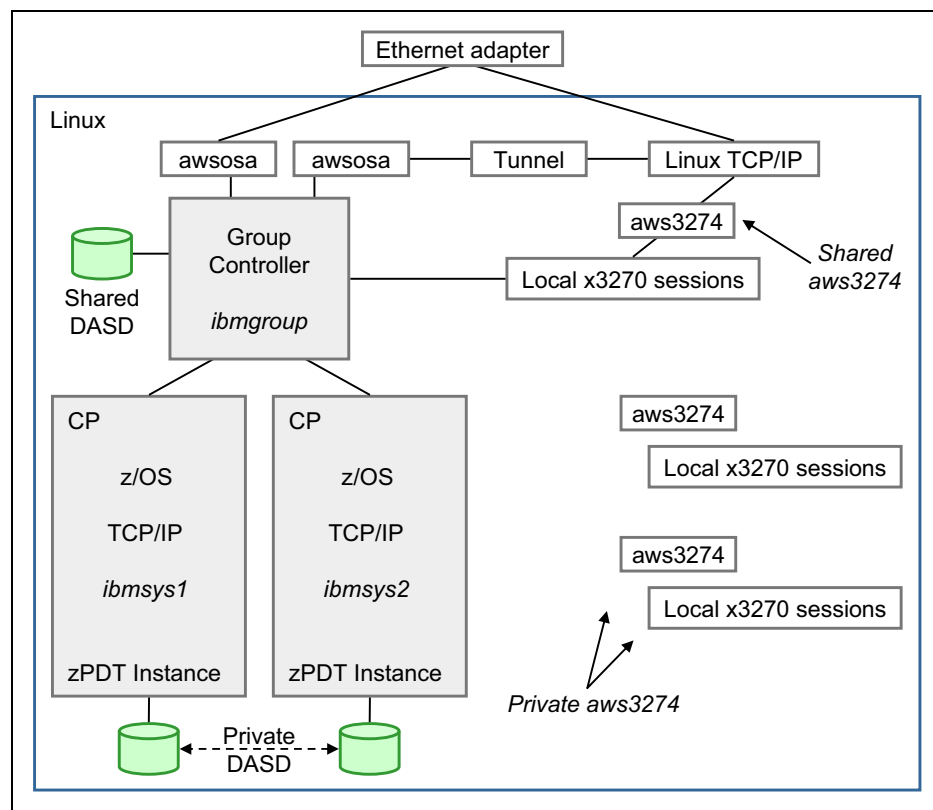


Figure 10-3 Shared emulated I/O

The Linux userid associated with the group controller must have the proper path information set in the `.bashrc` file, just like that for the userids associated with each instance. All the userids involved (the group controller and the instances) must be in the same Linux group; this is group `ibmsys` in our examples. All of the emulated volume files must be readable and writable (possibly via the `groupid`) by all the userids involved.<sup>2</sup> For our example, assume we have three userids defined (`ibmgroup`, `ibmsys1`, and `ibmsys2`) and all are in group `ibmsys`. We can define three devmaps, as follows:

```
/home/ibmgroup/group1
[system]
members ibmsys1 ibmsys2          # userids 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 --unitadd=0
device 401 osa osa --unitadd=1
device 402 osa osa --unitadd=2
device 403 osa osa --unitadd=3
device 404 osa osa --unitadd=4
device 405 osa osa --unitadd=5
device 406 osa osa --unitadd=6
device 407 osa osa --unitadd=7

/home/ibmsys1/aprof1
[system]
memory 800m
3270port 3270
processors 1
group ibmgroup                  #userid of the group controller

[manager]
name aws3274 4455
device 0700 3279 3274 mstcon
device 0701 3279 3274

/home/ibmsys2/aprofSB
[system]
memory 1000m
3270port 3271
processors 2
group ibmgroup                  #userid of the group controller

[manager]
name aws3274 5544
```

<sup>2</sup> This is controlled by normal Linux permission settings for each file. For example, the command `chmod g+w /z/*` could be used to make all the files in directory `/z` writable by members of the current group.

```
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 userid associated with each instance in the group.
- **group cntlname** is used in each instance and specifies the Linux userid associated with the group controller.

TN3270e sessions are directed to the desired 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 using shared OSA. For example, each instance might use an OSA interface at addresses 400-403. Each instance might start unit addresses (specified in the devmap) at address zero. (Multiple guests under z/VM, in a single instance, must manage the addresses properly. Do not confuse multiple guests under z/VM with multiple instances.)

Only DASD (CKD or 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. Notice that the two instances in the previous example have different 3270port addresses. We elected to not use shared 3270 definitions in this example.<sup>3</sup> No DASD is defined for the instances in this example; the instances will share the DASD defined for the group controller.

All sharing instances use the same addresses (device numbers) for the shared devices. There is no provision to have different addresses (for different instances) for the same shared device.

If a zPDT instance operates under the group controller, then any OSA devices might be shared devices, managed by the group controller, or each instance may 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 address(es) for each instance.

Standard operating rules still apply, of course. We cannot IPL the same z/OS system into two instances at the same time.<sup>4</sup> In our small example, we have two z/OS systems (the second one is on the SARES1 volume provided with the AD-CD package). In the absence of shared ENQ functions<sup>5</sup>, you must manage any active data set sharing. The zPDT system correctly emulates disk RESERVE and RELEASE functions and these protect VTOC, catalog, and some other updates in the normal z/OS manner.

Startup for our controller and two instances, working from the Linux desktop, might go as follows:

```
(login as root; open a terminal window)
# xhost +                               allow multiple users to start x3270
```

<sup>3</sup> An example using a single 3270 port number is given later in the text.

<sup>4</sup> This statement ignores situations where the use of different PARMLIB members allows an IPL of the same z/OS in multiple LPARs or instances. This involves separate paging, spooling, and various VSAM data sets for each LPAR or instance. The z/OS AD-CD system used for many of our examples is not configured for this type of use.

<sup>5</sup> 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.

# su ibmgroup	<i>work as group controller</i>
\$ cd /home/ibmgroup	
\$ awsstart group1	<i>start group controller</i>
(startup messages)	
(open another terminal window)	
# su ibmsys1	
\$ cd /home/ibmsys1	
\$ awsstart aprofl	<i>working as ibmsys1</i>
(startup messages)	<i>ibmsys1 instance</i>
\$ x3270 -port 3270 mstcon@localhost &	<i>working as ibmsys1</i>
\$ x3270 -port 3270 tso@localhost &	<i>working as ibmsys1</i>
\$ ip1 a80 parm 0a8200	<i>working as ibmsys1, IPL z/OS</i>
(open another terminal window)	
# su ibmsys2	
\$ cd /home/ibmsys2	
\$ awsstart profSB	<i>working as ibmsys2</i>
(startup messages)	<i>ibmsys2 instance</i>
\$ x3270 -port 3271 mstcon@localhost &	<i>working as ibmsys2</i>
\$ x3270 -port 3271 tso@localhost &	<i>working as ibmsys2</i>
\$ ip1 a90 parm 0a90sa	<i>working as ibmsys2, IPL z/OS</i>

In this example, we used a different terminal window to start the group controller and each z/OS instance. This allows us to send commands (such as **awsstop**) to the appropriate application later.

## 10.5 Additional shared functions

The previous section outlines the key shared device usage rules, as they apply to DASD and OSA devices. The group controller can also 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 has specified a port address for TN3270e connections. Each instance inherits the complete set of 3270 device definitions (700, 701, 702) but not the 3270port address. That is, each instance has a 3270 at address 700, 701, and so forth. If a user starts a TN3270e session connected to the 3270port number on Linux, the user has several options:

\$ x3270 -port 3270 localhost &	<i>Example 1</i>
\$ x3270 -port 3270 ibmsys1@localhost &	<i>Example 2</i>
\$ x3270 -port 3270 ibmsys2.701@localhost &	<i>Example 3</i>

```
$ x3270 -port 3270 ibmsys1.mstcon@localhost & Example 4
```

In Example 1, the desired 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.<sup>6</sup> This selection menu is illustrated in Figure 10-4.

In Example 2, the first available 3270 in the `ibmsys1` instance is assigned. (The *instance name* corresponds to the Linux userid that started the instance.) Examples 3 and 4 specify both an instance name and the 3270 device identifier.

```
*** Welcome to the zPDT selection menu ***
Please select the member to connect from the list below
or type in the member and/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

As a reminder, 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. This involves an `inetd` service that automatically detects which 1090 instances are running and constructs a selection menu 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`) will have 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. This is not very meaningful for a small device list as shown here, but might be more meaningful for longer lists.

<sup>6</sup> 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.





## The awscmd command

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

Configuration is similar to other device managers:

```
[manager]
name awscmd 20
device 560 3480 3480
```

The CUNUMBR (which is 20 in this example) is an arbitrary hexadecimal number (up to four hex digits) that cannot duplicate the CUNUMBR used with any other device manager. Typically only one device is used. The device type can be 3420, 3422, 3480, 3490, or 3590; these are the tape device types emulated by zPDT. The device number (560 in the example) must match a corresponding device type in your z/OS IODF. (Any device number may be used with z/VM.)

The intended operation is as follows:

1. A rewind is issued to the device.
2. The desired Linux command (expressed in EBCDIC) is written to the device.
3. Any stdin data to be used by the Linux command is written to the device.
4. EBCDIC to ASCII translation is done automatically; binary data is not possible.
5. A tape mark is written to the device.
6. At this point, the awscmd device manager submits the command (and data) to Linux through a shell that does not appear in the Linux window. The Linux current directory for the command is the directory that was used to start zPDT.
7. When the awscmd function completes there are four files on the pseudo-tape device:
  - The command file that was submitted to Linux (with redirection operands that were automatically added by awscmd)

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

## Restrictions

The command you send to Linux cannot include any redirection (< or > characters), asynchronous indicator (ampersand (&) character), or pipe (“|” or vertical bar character). The pseudo-tape device appears to be busy while Linux is executing 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 CMS user as device 28F:

```

/* CMS REXX script to execute 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 could 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 5300PT.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 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. You might want to examine the sample program we have provided in 11.2.1, "Sample z/OS program for awscmd" on page 210. 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 this 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 be enabled for BLP<sup>1</sup> processing. This can be done by changing the JES2 startup parameters or (for the duration of an IPL) by entering the following command on the MVS console:

```

$T JOBCLASS(A),BLP=YES          (you might want to use jobclass other than A)

```

We then mount a tape on the pseudo-tape drive for continued use. This avoids having to respond to a mount message every time the sample program is run.

```

MOUNT 560,VOL=(NL,123456)

```

<sup>1</sup> BLP means Bypass Label Processing; this prevents z/OS from testing for a standard label on the tape.

The MVS **mount** command followed by the zPDT **ready** command provides the necessary setup:

```
$ ready 560
```

An example of using the sample program might be this:

```
//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 the **awscmd** command. Nothing is actually written to the device. The job output (viewed from JES2 spool using 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.xxetc
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 actually executed 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 shown here.

A second example, using SYSIN data to create a new Linux file, could be this:

```
//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
/*
```

This example creates Linux file `my.new.file` (in the Linux directory used to start zPDT) with the indicated lines in the file. A fully qualified Linux file name could be used with either of the examples. The output for the second example would list all the data lines with the **COMMAND** output and then list them again for **STDOUT** output. (This is because **tee** writes all the stdin data to stdout. The **tee** command appends the new data if the specified file already exists.)

### 11.2.1 Sample z/OS program for awscmd

The following listing is a z/OS batch program, **AWSCMD**, that exploits 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 is not intended to illustrate the best programming techniques, but it should be fairly readable. Errors result in WTO messages; this

is a poor design for significant applications but it should be reasonable for many zPDT environments. The program includes a two-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 WTM operation until the Linux command completes.)

The following listing contains a minor update from previous versions: 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 via AWSCMD and read the result
* JES2 must allow BLP for the job class that is used
*      $T JOBCCLASS(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, if not wanted)
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 my 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'     Dis OPEN work?
      BZ NOINPUT            If not, branch
      MVI PARMFLG,X'FF'     Remember to use SYSIN
* Open pseudo0-tape device and SYSPRINT
A OPEN STAPE0              It should have BLP on statement
      TM STAPE0+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 (it might not be necessary but does not hurt)
B MVI SECB,X'00'           Zero my ECB
      LA 1,SCREW            address of CCW
```

```

        ST    1,SIOB+16      Place @(CCW) in IOB
        EXCP  SIOB           Execute 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 my ECB
        LA    1,SCCWRITE     Get @(write CCW)
        ST    1,SIOB+16      Put it in the IOB
        EXCP  SIOB           Execute 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 my 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              If not, branch
*
* WAIT AN ARBITRARY TWO SECONDS FOR LINUX
*
*E       STIMER WAIT,BINTVL=SEC2
*
* Rewind the pseudo tape
E        MVI  SECB,X'00'     Zero my 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 my 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 my 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 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 returned from Linux
J      MVC   BUFFER2(80),BLANKS ux

```

```

MVC    BUFFER2+0080(80),BLANKS
MVC    BUFFER2+0160(80),BLANKS
MVI    SECB,X'00'          Zero my 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 (STAPE0,,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
STAPE0   DCB   DSORG=PS,MACRF=(E),DDNAME=TAPE
PRINT    DCB   DSORG=PS,DDNAME=SYSPRINT,MACRF=(PM),LRECL=132,      X
          DCB   BLKSIZE=13200,RECFM=FB
SYSIN     DCB   DSORG=PS,DDNAME=SYSIN,MACRF=(GM),LRECL=80,        X
          DCB   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(STAPE0)        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
/*

```





## Minor z/OS notes

This chapter is not intended as a general z/OS guide, or as a guide to the AD-CD systems. However, several common questions or problems are discussed here.

### 12.1 How much storage for z/OS?

Starting with z/OS 2.3 a minimum of 2 GB System z storage (memory) is required<sup>1</sup> when running z/OS under zPDT. (A minimum of 8 GB is required on other systems.) This does not mean that 2 GB is a practical system; it very much depends on what you are running with z/OS. The 2 GB size might be practical for a simple TSO system, with a few users and with no major subsystems running. It is probably not a practical system for someone using WebSphere, IBM DB2®, and significant Java applications. As always, paging levels in z/OS and the base Linux should be monitored. We observe that most zPDT users have at least 8 GB for z/OS and some are much larger.

### 12.2 DEFLATE and zEDC emulation

It is important to understand the terminology involved. The zEDC function available with earlier System z systems and with zPDT GA9 is properly named “zEDC Express” or “zEDC Express adapter” and the new function in z15 systems is simply zEDC. To be clear, here we use the term “zEDC (DEFLATE)” to describe the new z15 facility.

IBM z15 systems provide the zEDC (DEFLATE) facility to replace the zEDC Express adapter available in earlier systems; zEDC Express adapters are not usable with z15 systems.

The initial zPDT GA10 release does not provide the zEDC (DEFLATE) facility. This release continues to provide zEDC Express adapter emulation. zPDT intends to provide the zEDC (DEFLATE) facility in a GA10 fixpack; the same fixpack will disable zEDC Express adapter emulation. The following text (written for zPDT GA9) also applies to zEDC Express adapter emulation in the initial zPDT GA10 release.

<sup>1</sup> “Required” is perhaps too strong; a console message is produced if the system is below the 2GB (or 8GB) size and the operator can elect to continue operation.

## 12.2.1 zEDC adapter emulation in earlier zPDT releases

zPDT GA9 provides limited zEDC emulation for z/OS. The limitations are these:

1. The compressed (“deflated”) data produced by the emulated zEDC is likely to be a different size than what would be produced with a hardware zEDC feature. The emulated zEDC compressed files are often smaller than those produced with the “real” hardware.
2. Data compressed with the zPDT emulated zEDC cannot be uncompressed (“inflated”) by a hardware zEDC feature. However, data compressed by a “real” zEDC can be inflated by the zPDT emulation.
3. Use with z/VM (or z/OS running under z/VM) is not supported.

Other than these limitations, the zPDT zEDC functions in the same way as a hardware zEDC feature on a large z System. This includes the automatic deflation and inflation during routine data use by z/OS, assuming DFSMS is properly set up. In general, the only time actual deflated data is seen is when dumping data or volumes with ADRDSSU or a similar product. The second limitation means that data dumped with ADRDSSU (or a similar product) cannot be moved to a “real” zEDC system and inflated there. Otherwise, zEDC usage is the same as it would be on a large z System.

To use zEDC the devmap needs a stanza such as the following:

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

The “120” in the example is a pchid number, the “0c0” is a function identifier, and the “3” is a virtual function number. The pchid and function identifiers are 3 or 4 arbitrary hex digits and the virtual function number must be in the range 1 to 15. These values are used internally in zEDC emulation but have no connection to any other configuration parameters. If more than one emulated zEDC is defined, the pchid and function identifiers must differ.

We found that the zEDC does not need to be included in the IODF; z/OS discovers it automatically. To use zEDC, a software feature must be enabled in PARMLIB member IFAPRDxx, as follows:

```
PRODUCT OWNER('IBM Corp')
NAME('z/OS')
ID(5650-ZOS)
VERSION(*) RELEASE(*) MOD(*)
FEATURENAME(ZEDC)
STATE(ENABLED)
```

We also specified PARMLIB member IQPPRM00 as follows, and added an IQP=nn statement to the relevant IEASYSxx member:<sup>2</sup>

```
ZEDC,MAXSEGMENTS=4,DEFMINREQSIZE=4,INFMINREQSIZE=16
```

Once started, z/OS commands **D IQP**, **D PROD,REG,FEATURENAME(ZEDC)**, and **D PCIE** can be used to verify that z/OS has detected it correctly.

---

<sup>2</sup> These are default values, but we specified them for completeness.

## 12.3 Maintenance for AD-CD z/OS systems

Users of the AD-CD z/OS systems have long requested a method of downloading PTFs. A method is in place to allow bulk downloading of PTFs in a format somewhat equivalent to the “PUT tapes” that were formerly used for z/OS maintenance. (These are no longer in tape format, but the terminology remains. PUT stands for Program (or product) update tape.<sup>3</sup>) We informally use the term “PUT” here, but only to refer to the material described in this section. These downloads are not provided by zPDT; they are provided by the AD-CD team. In principle, anyone who has authority to download the z/OS AD-CD volumes from the IBM Dallas site can also download the material described here.

This optional material can be used for software maintenance for the products in the z/OS AD-CD package. This material is for experienced z/OS systems programmers and is not intended for casual AD-CD users. Most AD-CD users will elect to obtain z/OS product maintenance through the periodic AD-CD releases as they have in the past. This facility might not be appropriate for all zPDT users for two key reasons:

- ▶ The PUTs are large, typically in the range of 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 the material can be handled by the user and by SMP/E.

Each PUT is divided into several approximately 1 GB files on the Dallas site. These files would normally be concatenated for use by z/OS.<sup>4</sup> PUTs have names such as PUT1909 representing PUT number 9 for 2019.

The Dallas site is at this address:

<https://dtsc.dfw.ibm.com/rexx/pgbd?/web/httpd1/pagebuilder/adcdzosmaint.dat>

A user name and password are required; this user name is unique to the Dallas download site and must be authorized for AD-CD z/OS downloads. The Dallas group intends to maintain eight PUT levels on this site, but circumstances might vary this number. Navigating this site provides selection of various PUT levels, such as PUT1811 for example. Selecting a PUT level lists the download files for that level, as in this example:

```
‘HTTPD1.fz206.f140.PUT1812.TRS10.BIN’
```

These can be downloaded through your browser download function. We suggest you rename the files when downloading them.<sup>5</sup> As an example, one recent PUT file contain 999 PTFs in 78 million records, and is typical of recent PUTs. The large size of these files is usually due to very 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 given 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.

<sup>3</sup> “PUT tape” is redundant, because it means *program update tape*, but the terminology is often used. “PUT” and “PUT tape” are used interchangeably.

<sup>4</sup> Another option is to combine the files into a single DSNTYPE=LARGE data set.

<sup>5</sup> 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 best to avoid this complication.

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 we ran an AMATERSE job to uncompress the data set and assign it a logical dataset 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) are RECFM=FB, LRECL=80. The job we use 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
```

We use FTP to transfer a file from Linux to OGDEN.PUTUP (which was allocated with 1500 cylinders, RECFM=FB, LRECL=1024); we then alter the AMATERSE job to have the right 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 and so forth.

## 12.4 Java performance

z/OS Java performance might be a problem under zPDT. The basic cause lies with the very heavy system overhead (CPU and I/O) involved when starting z/OS Java applications, especially applications that load many classes. The following notes might help improve performance in some cases; the improvements can be substantial in some cases.

### **General base system**

Even simple Java applications tend to perform heavy zFS I/O during startup, as the Java virtual machine loads the class libraries from the disk archives. If this I/O is contained mostly in the zFS cache and/or the Linux disk cache, general performance is improved. In other words, more PC memory can help. There are no specific Linux parameters for tuning the disk cache. The Linux **free -m** command can be used to monitor PC memory use, although interpretation of the data produced can be a little difficult.

The raw performance of the base PC is an obvious concern, as is anything that can detract from this such as virtualization at the Linux level or at the z/VM level.

### **Basic Java tuning**

There are a variety of approaches to tune the JVM start-up, depending on your application characteristics, deployment pattern and performance metrics. JVM features, like Quick Start and Sharedclasses caching, along with tuning of garbage collection settings, can improve [https://jazz.net/wiki/bin/view/Main/SetupBuildForPerformance#Java\\_startup\\_time\\_and\\_ant\\_tuning](https://jazz.net/wiki/bin/view/Main/SetupBuildForPerformance#Java_startup_time_and_ant_tuning)

Current Java implementations use SIMD ("vector") instructions to accelerate string, crypto and arithmetic operations on "real" hardware. The performance of these instructions varies with zPDT and, in some cases, Java performance might be better if the SIMD instructions are not used. This can be done by including the statement **-Xjit:disableAutoSIMD** in the jvm.options parameters used for your Java application.

DBB release 1.03 (and later) contain a *persistent JVM* that can substantially improve performance. Verify that you are at least at level 1.03.

### ***RTC usage***

We have noted that LAN performance between an RTC client system (that is, a separate PC) and the RTC server (z/OS) can be very dependent on LAN speed -- much more so than you might expect. We suggest you review the material in 7.6, "Performance problems" on page 133.

### ***RESOLVER timeouts***

With earlier z/OS Java levels an unexpected Java performance problem can occur if you enable the z/OS RESOLVER, specifying one or more external name server IP addresses, and if the RESOLVER fails to connect to the name servers. In this case any Java application that is started (including javac) hangs until the RESOLVER experiences a time out trying to connect to the name servers. This 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 simply not running the RESOLVER, or by not specifying external IP addresses of name servers for it, or by verifying it connects to a working name server. With IBM SDK for z/OS, Java Technology Edition, Version 8 SR5 FP35 (8.0.5.35) or newer, the JVM option **-XX:-ReadIPInfoForRAS** will instruct the JVM to not query IP information during start up, and will bypass this RESOLVER issue

### ***zOSMF***

Recent z/OS AD-CD systems automatically start zOSMF. This startup, using default parameters, takes considerable time (both CPU and elapsed). If you do not want zOSMF started automatically, you can take two approaches:

- ▶ Prevent any zOSMF startup. To do this, edit member IZUPRMNO in the ADCD PARMLIB to read as follows:  

```
AUTOSTART(CONNECT)
AUTOSTART_GROUP('NONE')
```
- ▶ Allow zOSMF to connect to a running zOsMF elsewhere in your sysplex (if it exists), but do not permit it to start on your current z/OS. To do this, use PARMLIB member IZUPRM00

In both cases, edit the IEASYSxx member(s) that you use in the ADCD PARMLIB, and change the IZU=xx parameter to IZU=NO or IZU=00.

Additional Java tuning options specific to zOsMF are in the following OMVS file:

```
/var/zosmf/configuration/servers/zosmfServer/jvm.options
```

This file is in ASCII. Within **oedit** the commands **SOURCE ASCII** and **RESET SOURCE** can be used to work with ASCII data.<sup>6</sup> With **obrowse** the commands are **DISPLAY ASCII** and **DISPLAY RESET**.

## **12.5 Excessive Health Checker messages**

The Health Checker is started automatically under z/OS 2.1 and later version. It is a useful tool, but it might check for details that are not relevant to zPDT users. As an example, when running a parallel sysplex system<sup>7</sup> we might see the following message on the MVS console:

<sup>6</sup> The equivalent commands for OBROWSE are **SOURCE ASCII** and **DISPLAY RESET**.

<sup>7</sup> The parallel sysplex system is described in IBM Redbooks publication SG24-8386.

```
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 health check as follows:

1. In the AD-CD z/OS 2.4 system IBMUSER has only *read* access to the Health Checker controls.<sup>8</sup> 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
```
2. Go the SDSF primary option menu and select option CK to access the Health Checker. Scroll through checks looking at the *Status* column on the right of the screen. This column contains text such as SUCCEs, 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.
3. 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.

## 12.6 z/OS spin loop timeouts

A z/OS spinloop might time out and produce an S071 ABEND. This 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 very rare. The most common reasons for SPINLOOP problems are an overcommitted virtual environment, disk cache writes when very large PC memory is used, or very high I/O rates from multiple application tasks. In rare cases, a situation with Hyper-Threading has produced SPINLOOP delays. Frequent messages indicate a need to examine your operational configuration carefully.

## 12.7 PER traps

The use of PER traps (typically with the SLIP command) causes zPDT to stop using its JIT (Just In Time interal compiler) while PER is active. This can result in much slower zPDT operation.

## 12.8 Larger 3270 display

The x3270 terminal emulator can be started with an optional parameter, as follows:

```
$ x3270 -port 3270 -oversize 133x60 localhost &
```

---

<sup>8</sup> It appears that user ADCDMST has *update* access to the relevant IBM RACF profile. You can log on with this userid and skip the RACF change.

This produces a 3270 window with 133 columns and 60 lines. (Other sizes may be specified; this is simply an example.) Basic TSO does not use the “extra” window space. ISPF can use it if max is specified for the window format in the ISPF option 0 panel.<sup>9</sup> (ISPF does not use the extra width unless a data set being displayed or edited has records that can use the extra width.)

Another method is to define a model 5 3270 terminal:

```
$ x3270 -model 3270-5 localhost:3270 & (ampersand not needed if in devmap)
```

However, the “standard” 3279-5 has only 27 lines.

## 12.9 z/OS disk STORAGE volumes

In some circumstances, z/OS functions use STORAGE volumes<sup>10</sup> for disk allocations. The distributed z/OS AD-CD systems generally have only a single STORAGE volume, usually named xxSYS1. For anything beyond trivial z/OS usage, we *strongly recommend* that *at least* one additional 3390 volume should be defined and mounted 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
D2SYS1,0,0,3390      ,Y
GEORGE,0,0,3390      ,N      <==we added this
WORK* ,0,0,3390      ,N      <==we added this
```

The format is fixed and the spaces shown are important. Briefly, the parameters for volumes are:

1. The volume serial number, in six columns. An asterisk matches any volser with the specified characters.
2. The first ‘0’ is the mount attribute and is normally set to ‘0’ unless there are complex mount circumstances.
3. The second ‘0’ is the use attribute. This 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.
4. The device type, in eight columns, left justified and padded with blanks.
5. An indicator whether z/OS should issue a MOUNT request for the volume. The value ‘Y’ indicates that z/OS will request this volume to be mounted at IPL time if it is not already mounted. Generally, ‘N’ is appropriate for local STORAGE volumes.

The placement of SVC dump datasets can create a problem because they typically go to STORAGE volumes and, by AD-CD default, this is the xxSYS1 volume.<sup>11</sup> An MVS command can be used to direct SVC dumps to new volume(s); the command would be similar to **DUMPDS ADD,VOL=(volser,volser,volser)**. Much more sophisticated controls are available through SMS functions, but these require administrative work to create the desired environment.

<sup>9</sup> You might need to scroll down in the ISPF option 0 panel to see the appropriate options.

<sup>10</sup> Generally, a STORAGE volume is used when creating a permanent dataset and no volume is specified, although the exact details involved can be a little more complex.

<sup>11</sup> You can look for data sets with names such as SYS1.ADCD.DMPnnnnn and delete them (assuming you are not working on a problem that involves these dumps). The exact data set name pattern for SVC dumps is set in the appropriate COMMNDxx member in PARMLIB.

## 12.10 Stand-alone z/OS dump

The z/OS stand-alone dump (SAD) is a program that is started (IPLed) from tape or disk. It does not run under z/OS. However, it assumes that z/OS is present in z System memory at the time the stand-alone dump is started. The stand-alone dump program is sensitive to the release of z/OS that is being used *and must match the z/OS release*. A new version of the stand-alone dump program should be generated whenever a new release of z/OS is installed.

This section describes a simplified use of stand-alone dump based on the AD-CD z/OS system release 2.4, but the comments should apply to any recent z/OS release. The dump program is placed on an “IPLable” disk volume and the dump output data *must* be directed to a different disk volume. The stand-alone dump program (and dumped output) can be used with tape volumes, but this is not further described here.

### Disk volumes

Two disk volumes are referenced here. One is for the dump program itself, which is relatively small (about 100 tracks). This volume also contains IPL text to start the stand-alone dump program. This volume must be mounted as PRIVATE to run the dump generation job, but can be in any mount status when IPLing the dump program. An additional rule is that the SAD IPL volume cannot contain a paging dataset for the system being dumped.

Many z/OS AD-CD users place the stand-alone dump program on a system volume, such as A4CFG1. For our example, we created two new 3390 volumes: DUMPPP (200 cylinders) at address AD0 and DUMPDD (3390-9) at address AD1. The stand-alone dump *program* volume (AD0 in our example) is not sensitive to its address.

The second volume is for the dump itself.<sup>12</sup> A stand-alone dump can be large: hundreds of cylinders up to many thousands of cylinders. In our example we assume a suitable volume is mounted at address AD1. Before using the stand-alone dump program, *you must create the dump output data set on this volume using an IPCS or REXX utility function*. The stand-alone dump data volume is sensitive to the address (AD1 in our example) and must stay at the address you specify.

The dump program (that you IPL) *cannot* be on the same volume that is to receive the dump data.

### 12.10.1 Generating a stand-alone dump program

The following job generates the stand-alone dump program and writes it on volume LOCAL1. You later IPL from this volume when you want to take a stand-alone dump.

```
//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=DAD1
```

<sup>12</sup> The dump output can be directed to tape instead of a disk volume. Our discussion here concentrates on disk output.

```

                                END
/*
//PUTIPL   EXEC   PGM=ICKDSF
//IPLDEV   DD     DISP=OLD,UNIT=3390,
//          VOL=(PRIVATE,RETAIN,SER=DUMPPP)
//TRK0TEXT DD     DSN=&TRK0TEXT,DISP=(OLD,DELETE)
//SYSIN    DD     DSN=&DSFSYSIN,DISP=(OLD,DELETE)
//SYSPRINT DD     SYSOUT=*

```

Carefully check the JES2 output when the job ends to ensure that it completed correctly. This is a somewhat unusual job. 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. It will have data set 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 have already written IPL text on the dump program volume (DUMPPP in this example), there will be an operator message and reply needed before the IPL text is replaced. Remember that this volume (DUMPPP in the example) must be mounted PRIVATE when this job is run.

The error messages (or non-zero return codes) seen when attempting 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 will always have at least two disk volumes 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 you designate) must be deleted before you can try the job again.

## 12.10.2 Stand-alone dump output dataset

Output (“the dump”) from the Stand Alone Dump program requires a preallocated dataset. This can be created with IPCS or with a REXX program 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
Please enter the volume serial ....
DUMPDD                                (assume DUMPDD is at address AD1 in our example)
Please enter device type .....
3390
Please 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 dataset can be used for only one dump. To reuse it you must reset or clear it. This is done with the same AMDSADDD program, selecting a different option at the first prompt.

### 12.10.3 Operating a stand-alone z/OS dump

We assume you have been running z/OS and need to take a stand-alone dump for some reason:

```
$ stop all           (stop the CPs)
$ ip1 ADO           (assume volume DUMPPP is mounted at this address)
```

Wait about 10 seconds and press Enter on the 3270 console at address 700. This produces console messages similar to these:

```
AMD083I AMDSADMP: STAND-ALONE DUMP INITIALIZED. IPLDEV: OAD0 LOADP:
AMD001A SPECIFY OUTPUT DEVICE ADDRESS (1):      (press Enter)
AMD101I OUTPUT DEVICE: OAD1
          SENSE ID DATA: FF 3490 10 3490 40  BLOCKSIZE: 29,120
AMD011A TITLE=my 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
```

## 12.11 Moving 3390 volumes

The following text describes a generic method of moving 3390 volumes between z/OS systems (including z/OS on zPDT). Another method, using a client/server application provided with zPDT, is described in Chapter 15, “DASD volume migration” on page 289.

zPDT-emulated DASD volumes can be transferred to other systems in several ways. Sending a volume to another zPDT system is especially easy. The Linux file that holds the emulated 3390 volume can simply be copied.<sup>13</sup> Optionally, the copy could be compressed (with **gzip**, for example) for transmission. The transmission could be by FTP, by a USB thumb drive, by burning a CD or DVD, or by various other means. The key concept is that a large Linux binary file is being transferred.

Moving an emulated 3390 volume to (or from) a non-zPDT system is a little more complex, because it must be handled in a z System format instead of a Linux format. The traditional method is to dump the volume to tape (using the ADRDSSU program) and then restore the tape on the target system. This method can also be used with emulated tapes (in awstape format), provided that both the sending and receiving systems can use this format.

The following example assumes both systems cannot use awstape format (otherwise we would use the easier method of creating a dump tape in awstape format). We assume there is a network connection between the source z/OS and the target z/OS system. We also assume that the target system is a zPDT system, although this is not a requirement for the technique described. See Figure 12-1.

---

<sup>13</sup> Do this when z/OS is not running, of course.

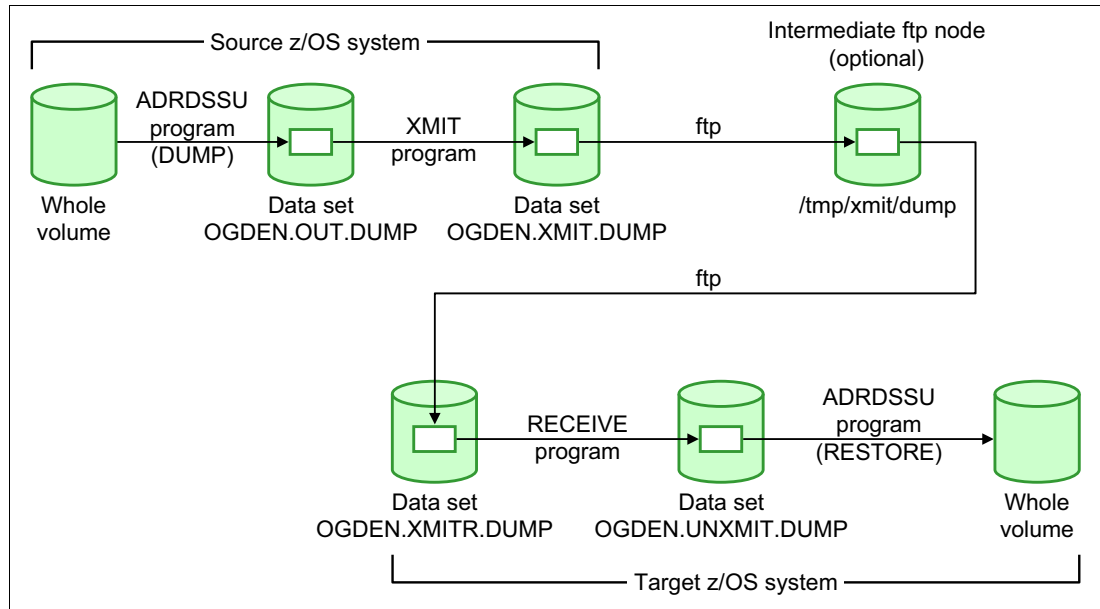


Figure 12-1 Overview of our example

## Preparation

We need z/OS disk space to hold a 3390 volume dump and to hold a reformatted volume dump. These can be large data sets. You might have sufficient space on existing z/OS volumes; we elected to create three new volumes for holding large temporary data sets. We did this using normal zPDT techniques. First we created three new emulated 3390 volumes using the `a1cckd` command (done while zPDT is not operational). The placement (/z directory), model (3390-3), and names of the files (TEMPnn) are all arbitrary.

```
$ a1cckd /z/TEMP01 -d3390-3
$ a1cckd /z/TEMP02 -d3390-3
$ a1cckd /z/TEMP03 -d3390-3
```

We then added these volumes to our devmap. The addresses specified (AA0, AA1, and AA2) are unused address that are known as 3390 devices for our z/OS. (That is, z/OS has these addresses specified as 3390 devices in the IODF it uses during IPL. The AAx 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
```

We then started zPDT and IPLed z/OS. During z/OS startup the new devices are recognized as uninitialized volumes and are 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; this is not required but is a good practice. After the volumes are initialized, they can be varied online to z/OS, using the MVS console:

```
vary aa0-aa2,online
```

## 12.11.1 Create 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 specified in the output DD statement might need to be adjusted, depending on the contents of the source volume. We next created another data set with specific DCB attributes.<sup>14</sup> (This step could be done 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 then 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 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 is meaningless in this example. Note that the **terse** program could be used instead of **xmit**.

It is important to understand the reason for the **xmit** step. In the general case, 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 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.

---

<sup>14</sup> These DCB attributes are used by XMIT.

Some FTP situations allow additional parameters such that the original block and record characteristics of a file are retained and the `xmit` step could be skipped. This can be done in a z/OS to z/OS FTP transfer. The method presented in this section assumes the more general case in which the FTP transfer does not retain the original block/record information.

### 12.11.2 Send dump to Linux

We then sent the xmit-formatted dump to Linux, using an FTP connection from Linux to z/OS. We used the zPDT tunnel facility for the connection in our example, but any TCP/IP connection to z/OS could 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 the FTP connection initiated from the Linux side. It could be done from the z/OS side, provided 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 using any technique suitable for a large Linux file. It could be compressed (using **gzip**, for example.) At this point 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.

### 12.11.3 Receive 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 is for the output of the TSO RECEIVE function. This last data set is then 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 using FTP. (If the file was compressed in Linux it must be uncompressed 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 zPDT z/OS system that we used to create the dump volume. In practice, this is likely to be a different z/OS system that might not be in a 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 then used TSO to reformat the dump into the original format 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. While perhaps a bit longer than absolutely necessary, we think the process shown here should work in almost any situation.

## 12.12 IODF Changes with zPDT

When working with a larger z System, IODF and IOCDS creation are almost always done at the same time, working with HCD. This 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 is a useful check for a larger z System, but it creates problems with zPDT.

A zPDT system does not use an IOCDS and does not understand many hardware CHPID details. Furthermore, typical zPDT configurations are not compatible with the normal HCD verification and processing we would use with a larger z System. At the time of writing, I/O device configuration for a z/OS system on zPDT consists of a *software only* IODF generation, followed by the creation of a matching devmap. Many z/OS users are not familiar with a

software only IODF and we provide an overview of the topic here; it is considerably simpler than a “normal” IODF.

Assume we want to add 15 OSA devices starting at address 410 and an OSAD device at address 41F. Using the IODF99 that is provided with the current z/OS AD-CD system<sup>15</sup>, we could proceed as follows:

```
HCD                                     (started via 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
```

At this point you should have a panel to 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).

This is followed by a panel to 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 . . . . 15
Device type. . . . . OSA      +
Serial number. . . . .
Description. . . . .
Volume serial number . . . _____ (for DASD)
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). This is followed by a panel allowing you to alter default device parameters; you should take the default options unless you have a particular reason for changing them. Press Enter. This is followed by a panel to associate esoteric names with the new devices; you might use this for DASD 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 410,15 in the list. Again select F11 (to add devices) and add a single OSAD device at address 41F, following the same steps used for the 15 OSA devices.<sup>16</sup>

When your new I/O devices have been added to the list, use F3 three times to return to the initial HCD menu. You then need to 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 may produce some warning messages, usually about esoteric
         tokens. Ignore these messages. F3 to exit the warning panel.)
```

<sup>15</sup> This example is based on the z/OS 2.2 AD-CD system, but later releases should be similar.

<sup>16</sup> OSA/SF (which uses an OSAD entry in the IODF) is no longer available with current z/OS releases. A partial replacement, QUERYINFO, is not supported under zPDT/

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 *****
. . . . .
***** *****BOTTOM OF DATA *****
```

Press F3 to exit this panel. The next panel allows you to 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 zPDT)
```

The next panel allows you to specify or revise these values; press Enter. This should produce the message PRODUCTION IODF SYS1.IODF77 CREATED. Use F3 several times to exit from 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:<sup>17</sup>

```
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 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 you IPL z/OS with the parameter:

```
$ ipl 0a80 0a8200          (The 00 corresponds to the LOAD00 member)
```

Assuming you are satisfied with the results, you will probably want to change all the LOADxx members that you use. You must also change your devmap to use the new devices you added to your z/OS system.

## 12.13 Many COBOL Compilations

If you run a large number of COBOL compilations every day you can substantially improve compiler performance by moving some compiler modules into LPA. For COBOL 6.2 we

<sup>17</sup> 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.

moved IGYCPGEN, IGYCSCAN, IGYDRV, IGYQCB, and IGYZQDRV to LPA and removed them from the IGY620.SIGYCOMP library.<sup>18</sup> It is necessary to remove them from this library because the library is used as a STEPLIB during compilation and a STEPLIB is searched before LPA. The named modules use about 1500 tracks and are program objects, not load modules. They cannot be simply placed in one of the LPA libraries; instead they can be handled by “LPA” statements in a PROG member in PARMLIB.

We did not try this technique with COBOL 6.3 (in the first z/OS 2.3 AD-CD release) but we assume the same improvements would be seen. The relevant library for COBOL 6.3 is IGY630.SIGYCOMP.

## 12.14 Local printing

There is often less need for *hard copy* printed output in today’s working environments, but it is sometimes needed. There are a variety of ways to approach this. The following material describes only one of these ways.

### Background

Basic z/OS printing is closely related to the hardware available on the original S/360 machines. The most common printer at that time 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 meant a full page 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, assemblers, compilers, system report programs, and so forth was designed to fit these pages. That is, 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. These were devices such as the IBM 3800, 3820, 3825, 3900, and so forth. These could accept a variety of paper sizes, but were most commonly used with “letter size” paper.<sup>19</sup> With proper programming, these printers can produce sophisticated output using many fonts and graphics 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 zPDT can emulate 1403 printers. This is all that is needed for printed output from utilities, compilers, and many existing applications.

### Using a PC printer

Our goal was to use a common PC laser printer and have utility output produced in the format just described: landscape mode, 66 lines per page, 132 characters per line. If the PC printer provides duplex printing (printing on both sides of the paper), this would be used. The flow is illustrated in Example 12-2.

Our tests used a Lexmark OptraS1250 and Optra L printers (both with duplex printing features). We have not tried the techniques described here with other printers, but we expect

<sup>18</sup> Actually, we simply renamed the modules in the SIGYCOMP library.

<sup>19</sup> The “letter size” (or A4 elsewhere) paper can be cut sheets or fanfold paper, depending on exactly which printer is being used.

the same or similar techniques could be used. However, remember that z/OS printed output typically contains separator pages, JCL listings and messages, and so forth; the smallest job usually has multiple pages of printed output. This may not be suitable for use with a small inkjet printer. We assume the use of a fairly heavy-duty laser printer for z/OS printing.

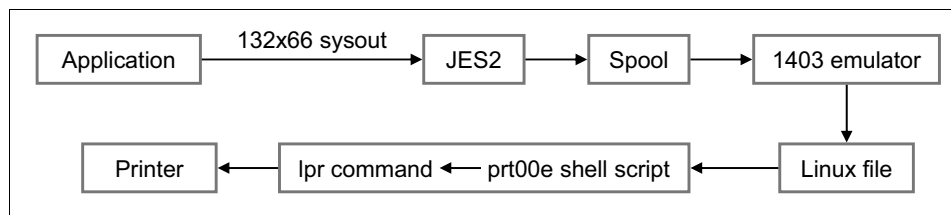


Figure 12-2 General flow for printing

## 12.14.1 Setup

We need to provide the setup for Linux, the zPDT devmap, a shell script, and JES2.

### Linux setup

We first configured our (very old) Lexmark Optra S1250 for Linux. This printer has a parallel input; our computer had no parallel ports. We purchased a USB-to-parallel cable and this provided the needed connectivity. We used YAST (running under openSUSE) to configure the printer.<sup>20</sup> A print queue named `optras1250` was created automatically by YAST. We verified that the printer worked by using commands such as:

```
$ lpr /home/ibmsys1/prof12          (to print one of our devmaps)
```

### Devmap setup

We added the appropriate 1403 definition to the zPDT devmap:

```
[manager]
name awsprt 4321 --windows
device 00E 1403 2821 /tmp/1403a
```

The `--windows` option is needed to place CR/LF characters in the output; without this option, NL characters are used and a PC printer may not be happy with NL characters. The output file name (`/tmp/1403a` in the example) is arbitrary. In our case, we did not expect much printed output and `/tmp` seemed a reasonable place to put it. The output file may also be assigned or changed with the `awsmount` command.

### Shell script

We created a Linux shell script named `prt00E` (the name is arbitrary) 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, written in octal.<sup>21</sup> The octal format was the most convenient for use in a shell script. If you have good script writing skills you can do this several ways. The particular control characters shown here are for the Lexmark printers we mentioned. Your printer may require different controls. If you are printing

<sup>20</sup> Red Hat Linux has a different configuration process, but the end results are about the same.

<sup>21</sup> CTL is shown as three lines here, but it is actually created as one long line containing 38 octal constants.

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 (via a *pipe* and `lpr`) to the queue we defined earlier. It sends the CTL string (using `echo`), then sends the print data from the output file we named in the `devmap` or `awsmount` command (using `cat`), and then sends the CTL2 string (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 a small line offset to better center the data. The only unique requirement is that the format must use *exactly* 66 lines per page in order to synchronize with the pages produced by the 1403 emulator.

In the following strings `\033` is the ESC character that is used to begin printer command strings, and this is shown as a bold-face **E** in the character equivalents of the string. The octal constants are the equivalent of the characters shown and the CTL string could be written with characters (except for the ESC byte).

The CTL commands are as follows:

OCTAL constant.....	Characters	Comment
<code>\033\105</code>	<b>EE</b>	Reset printer
<code>\033\050\163\060\160</code>	<b>E(s0p</b>	Use a fixed font,
<code>\061\066\056\066\067\150</code>	<b>16.67h</b>	with 16.67 inch character pitch
<code>\070\056\166</code>	<b>8.v</b>	with 8 lines/inch characteristics
<code>\060\163\060\142\124</code>	<b>0s0bT</b>	using upright, medium Courier
<code>\033\046\154\061\157</code>	<b>E&amp;llo</b>	Use landscape format
<code>\061\163</code>	<b>ls</b>	with duplex printing, long edge
<code>\065\056\064\143</code>	<b>5.4c</b>	5.4/48 inch line height
<code>\055\061\060\060\060\132</code>	<b>-1000Z</b>	line offset

The CTL2 commands are as follows:

<code>\014\033\105</code>	Force last page, reset printer
---------------------------	--------------------------------

The CTL string was produced after some experimentation. It works for our printers, but it may contain unnecessary elements. Notice that we hard-coded the file name (`/tmp/1403a`) in the shell script; more skilled users may want to make this a command-line variable.

## JES2 setup

Normal z/OS printing flows through JES2 and printers must be known to JES2. Recent AD-CD systems do not have a printer defined for JES2. We need to define a 1403 at address 00E for JES2. (We use device number 00E because it is the traditional address for a 1403 printer and because it is already defined in the AD-CD IODF.) Edit the AD-CD 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 AD-CD system) you will find commented lines similar to this. You can insert a new line, as shown, or convert the commented lines into active lines.<sup>22</sup> 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. (There are various ways to do the same task without the re-IPL.) We verified that the printer was online (`d u,,00E,1`)

<sup>22</sup> If you convert the commented lines into active lines, be especially careful with the `/*...*/` comment indicators; be certain you remove matching pairs.

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.

## 12.14.2 Operational technique

We then 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=*
```

After submitting this job, you should see z/OS console messages about jobs 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 is file /tmp/1403a in Linux. Additional output (from multiple jobs) is simply added to the file. The emulated printer cannot distinguish where one job ends and the next begins; the JES2 separator pages are needed for this.

At some point you can stop the JES2 printer (\$PPRT1) and print the accumulated output under Linux. (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
```

You then run the shell script:

```
$ cd /home/ibmsys1          (Directory containing the shell script)
$ ./prt00E                  (Execute the shell script)
```

The printer begins printing output. Notice that the output includes all the job separator pages 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; same name)
$ awsmount 00E -m /tmp/1403a (Connect new output file)
```

We deleted the output file (assuming we do not want to print the same jobs again). We then re-created the same file (because the name is hard-coded in the shell script) and “mounted” it on the emulated printer, ready for more output.

If you stopped the JES2 printer, you need to start it again (\$SPRT1).

## 12.15 SYS1.LOGREC full

Maintaining SYS1.LOGREC<sup>23</sup> is a normal z/OS system programmer’s task. There is nothing unique to zPDT or the AD-CD distribution. Production installations, using larger z System machines, often keep ordered histories of LOGREC data and study any new material in

<sup>23</sup> SYS1.LOGREC is the traditional name for this dataset; however, installations can change this. Recent z/OS AD-CD systems use SYS1.S0W1.LOGREC to allow a logical step to using the name SYS1.S0W2.LOGREC for another z/OS in a sysplex configuration.

LOGREC. It includes data about hardware and software failures, IPL statistics, volume use statistics, and so forth.

Most zPDT users simply ignore SYS1.LOGREC until they receive messages that it is full. These messages do no harm, but it is a good idea to clear LOGREC when such messages are received. There is at least one job in the AD-CD libraries to do this, but the job name (and library name) may change from time to time. The following is a job to clear SYS1.LOGREC. The particular format shown here is quite old and should be used exactly as shown.

```
//BILLCL JOB 1,OGDEN,MSGCLASS=X
// EXEC PGM=IFCEREPI,PARM='CARD'
//SERLOG DD DISP=SHR,DSN=SYS1.SOW1.LOGREC
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//EREPT DD SYSOUT=*,DCB=BLKSIZE=133
//ACCDEV DD DUMMY
//TOURIST DD SYSOUT=*,DCB=BLKSIZE=133
//SYSIN DD *
    SUM
    ACC=Y
    ZERO=Y
/*
```

You may find slightly different jobs that perform the same function and any of these should be acceptable. Never attempt to “clear” SYS1.LOGREC by simply deleting and reallocating it, or by removing records with a text editor.

You can edit IEASYSxx members (in PARMLIB) to say LOGREC=IGNORE to avoid the LOGREC full problems.

## 12.16 Lost MVS console

MVS does not like to lose its operator console (and this is not unique to zPDT operation!) If you accidentally close the TN3270e session that contains the MVS operator console, you might try the following recovery.

First, simply try to re-establish the session. This might be easier if the MVS console has an LName 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 and you cannot make a “new” connection to it. You can force the console session to completely disconnect by this command in a Linux window:

```
$ awsmount 700 -d (assuming your MVS console is address 700)
```

You might then be able to connect the TN3270e session to address 700. You need to have the TN3270e session connected before proceeding with additional recovery.

When MVS lost the console it probably started issuing messages in the Linux window used to start zPDT. These are “HMC hardware console” messages. You can attempt to reactivate the MVS console on 700 (assuming you have a TN3270e connection to 700) as follows:

```
$ oprmsg v 'cn(*),activate' (activate the "hardware console" 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 might not always work. Also, it might produce a console in 3270-2 (24 lines) mode, but this is better than no console. The single quotation marks in the commands are needed to prevent the Linux shell from directly using the parenthesis characters in the commands.

## 12.17 Unable to start ISPF

When logging onto 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 uncataloged for some reason. This 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. Crashing z/OS at just the wrong moment might also do this.

You need to recatalog the profile data set. Assuming you are logging on as IBMUSER, the data set you want (in all recent AD-CD releases) is IBMUSER.ISPF.ISPPROF. You can log on with another userid (ADCDMST is convenient for this purpose) and recatalog the data set. The easiest way to recatalog the data set is to list the volume (ZCSYS1, for example) using ISPF 3.4 and to use a **C** line 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 (using basic TSO commands) and then start ISPF again. To do this, use the following TSO command:

```
READY alloc da('ibmuser.ispf.ispprof') f(ispprof) shr vol(zcsys1) unit(3390)
READY ispf                                (start ISPF again)
```

You need to use the volser that matches your current system, of course.

## 12.18 Customized Offering Driver (COD)

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 CBPDO. The COD, at the time of writing, included 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 containing documentation.
- ▶ A very wide range of addresses (device numbers) for 3390s and local 3270 consoles are included, with a smaller range of addresses for SNA 3270-X, 3590 (tape), OSA, CTC, and SCTC devices.
- ▶ TCP/IP is usable with some configuration work.
- ▶ Instructions are included for working as a z/VM guest.

Why would a zPDT user want to use the COD? The reasons might include:

- ▶ A desire to work with a more “basic” z/OS system than the AD-CD z/OS system. The AD-CD z/OS system contains a considerable amount of customization making it easier for

many people to use it. Systems programmer training might include starting with a much more basic z/OS level and the COD could be a more appropriate starting point.

- A simple practice platform before using the COD on a large Z system.

The COD is not part of zPDT deliverables and cannot be obtained through zPDT or AD-CD channels.

The following example is based on the SUSE Leap42.2 Linux 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 you are working with Linux userid *ibmsys1*. This is arbitrary, as is the directory name `/mnt/ibmdvd`; we use these to provide specific installation examples. We use **su** to work as *root*; you could use **sudo** instead if your Linux system is configured for it.

Before starting, create a Linux directory 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 these are used in our example. We created our emulated volumes in directory `/z`, but this is arbitrary, as are the Linux names we used to contain the volumes.

```
$ alckd /z/D9ECAT -d3390-9
$ alckd /z/D9ESY1 -d3390-9
$ alckd /z/D9ESY2 -d3390-9
```

We created a zPDT devmap named `devcod`, as follows:

```
[system]
memory 8G
processors 1          #You can specify 3 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 three 3390 volumes are installed in the same way; the following text illustrates installing the D9ECAT volume.

```
(insert the D9ECAT DVD and assume Linux will automount it.)
# su                                (switch to root)
# umount /dev/sr0                  (unmount the dvd; name may vary)
# mount -t iso9660 -o ro,map=0 /dev/sr0 /mnt/ibmdvd
# exit                             (leave root)
$ ls /mnt/ibmdvd                  (should see CAT)
$ ls /mnt/ibmdvd/CAT              (should see multiple files)
```

```

$ awsstart devcod                                (start zPDT)
$ ipl_dvd /mnt/ibmdvd/CAT/DFSMSDSS.INS             (upper case 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 screen.)
  (You should see a message to clear the screen. Use ONLY the 3270 clear
   operation at this point.)
$ 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 screen if requested.)
$ ADRY0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0
$ awsstop

```

We stopped zPDT between restoring volumes and unmounted the current dvd:

```

$ su
# umount /dev/sr0
# eject

```

We then 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 lower case if you prefer. Upper or lower case is critical for Linux commands but is usually 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 VTAM modtabs provided with the COD support only 24x80 sessions for VTAM and TSO. If we use default x3270 screens (43x80) it is easy to mistake a partly-filled screen (which can be scrolled forward) for the end of a list. The MVS console automatically adapts to the 43x80 screen size.

After starting zPDT, we proceeded as follows:

```

$ ipl 0170

```

After a few seconds the MVS console interaction (with the 3270 at address 00A1) went as follows:

```
IEA101A SPECIFY SYSTEM PARAMETERS FOR Z/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 our TSO terminal is online)
s vtam                                    (start vtam & 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 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 should produce a CustomPac Master Application Menu. The "P" option should start a normal ISPF session.

To shut down z/OS cleanly, logoff 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 CLPA or COLD start JES2.

## 12.18.1 TCP/IP connection

We wanted to establish a TCP/IP connection to the COD z/OS system. The COD contains many OSA definitions and we arbitrarily decided to use OSA addresses 0300-0302. The profile parameters for TCPIP are in TCPIP.SEZAINST, the VTAMLST parameters are in SYS1.VTAMLST, and the TCPIP 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):

- Add OSA definitions to our devmap (and restart zPDT):

```
[manager]
```

```

name awsosa 0006 --path=A0 --pathtype=OSD --tunnel_intf=y
device 0300 osa osa
device 0301 osa osa
device 0302 osa osa

```

- IPL the COD and start TSO (in a local 3270 session), as described above. We examined the default TCPIP 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 DEFAULT 10.1.1.1 ETH1 MTU DEFAULTSIZE
ENDRoutes
TCPCONFIG RESTRICTLOWPORTS
UDPCONFIG RESTRICTLOWPORTS
IPCONFIG NODATAGRAMFWD
START PORTA

```

- We altered SYS1.PROCLIB(TCPIP) so that the PROFILE DD statement points to member PROFILE2 instead of member PROFILE.
- We added member SYS1.VTAMLST(OSATRL1) as follows:

```

OSATRL1 VBUILD TYPE=TRL
OSATRL1 TRLE LNCTL=MPC,READ=(0300),WRITE=(0301),
          DATAPATH=(0302),PORTNAME=(PORTA),
          MPCLEVEL=QDIO
(Be certain the continuation marks are in column 72)

```

- We edited member SYS1.VTAMLST(ATCCON00) member as follows:

```
TSOAPPL,TCPAPPL,OSATRL1
```

- We then relPLed z/OS. After the startup process described above we entered z/OS commands as follows”

```

v 300-302,online           (Bring the OSA ports online)
s tcpip
s tn3270

```

At this point we could ping 10.1.1.2 from Linux. Starting a new x3270 session connected to 10.1.1.2 we received an UNFORMATTED SYSTEMS SERVICES prompt and could logon to TSO. (We needed to first logoff from the local 3270 session because we had only a single TSO userid.) When shutting down z/OS you should also stop TN3270 and TCPIP.

LAN configuration can be frustrating because there are so many details that must be exactly correct. The steps described here create a link to the base Linux TCP/IP; this is probably the

most basic TCP/IP configuration possible with the COD and zPDT. It is a good starting point before attempting to configure an external TCP/IP link.

## 12.19 WLM and AD-CD

The recent z/OS AD-CD systems are provided with a WLM service definition already installed. While there is nothing wrong with the supplied WLM definition, it might not be overly useful with typical 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 zPDT owner who has not previously worked with WLM.

WLM service definitions are created and used in the WLM coupling dataset(s).<sup>24</sup> It is possible, although not required, to then extract and write the WLM service definition to a PDS. A number of PDS members are created when doing this, and the data is not in a “human” format. The same PDS can later be installed in the coupling dataset (overwriting what is there) and then activated. At the time of writing the WLM coupling dataset is SYS1.ADCDPL.WLM.CDS01 (and CDS02); the PDS copy is ADCD.Z24A.WLM.

WLM definitions can be quite complex, but can also be fairly simple if only simple controls are needed and this is likely to be the case for a 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 WLM at any one time<sup>25</sup> and only one *policy* within that service definition can be *active* at any one time. However, you can work on an additional service definition (using ISPF) in the coupling dataset while a different service definition is being used.

We designed a service definition named BASEAD, having a single policy named BASEAD1, with the characteristics shown in the table.

Workload Name	Service Class Name	Imp	Service class goal	Classification
HIGH	FAST	1	90% velocity	LDAP LSFM STC TCP
TSO	TSO	2	90% in .25 seconds	TSO
CICS	CICS	3	80% in 1 second	EWLM CICS IMS
SERVERS	MEDIUM	3	50% velocity	CB ASCH Db2 DDF IWEB MQ SOM
BATCH	SLOW	4	Discretionary	JES

Table 12-1 Trivial WLM definition

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 trivial example. The “Imp” column is the “importance” specification for the service class. 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 work in this service class is expected to run at 90% of the processor speed. The “90% in .25 seconds”

<sup>24</sup> Coupling datasets are normally associated with sysplex operation, but the WLM usage is independent of whether or not a sysplex is being used.

<sup>25</sup> Only one service definition can be *installed* in the WLM coupling dataset at any one time. The keyword is “installed;” this means a policy within it can be *activated*. A completely different service definition can be concurrently built (using ISPF) in the same coupling dataset, but it is not “installed” at this point.

indicates that 90% of the transactions in this class should complete in .25 seconds or less. Discretionary means there is no specific goal and usually indicates the lowest performance class.

Our service definition is a little odd in that it places TSO importance above that of servers such as DB2, CICS, IMS, MQ, and so forth. 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; if this were not the case we would have placed all these in the SERVERS workload.

There are many IBM manuals, IBM Redbooks publications, and third-party documents discussing WLM parameters in great detail; we do not attempt to repeat any of this material. The purpose of the discussion here is to describe the mechanics of creating and using your own WLM service definition, using our trivial example. As with most z/OS administrative activities, there are many ways to go about it and we describe a very basic approach that can be used as a starting point for more complex work.

We proceeded as follows (using a recent z/OS AD-CD system):

- ▶ Decide on the basic design of your service definition, including initial workload names and service class names. You can expand these later.
- ▶ Create a small PDS to save your workload once you have defined it. Each service definition requires a separate PDS for saving it. We suggest 5 tracks, 5 directory blocks, LRECL 80, RECFM FB.
- ▶ Go to ISPF option M.12 to start a WLM interactive dialog. Select option 3 to create a new definition.
- ▶ 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. (PF3 twice to return to the main WLM menu.)
- ▶ Select option 2 to create a workload name. (PF3 twice to return to the main menu.) (Within the workload panel, use option 1 to create another new name.)
- ▶ After defining your initial workload names, use option 4 to define service classes. For each assign a service class name and the name of the corresponding workload that you previously created. Then position the cursor to the Action field and enter I (for Insert new period). This opens a small panel where you can select the a goal for this service class, such as a velocity percentage, an importance parameter, and a duration (to be used if there are multiple periods for this service class).<sup>26</sup> 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.
- ▶ After defining your workloads and service classes, use option 6 for Classification Rules. This displays a panel with about 17 predefined classifications. Do not attempt to create new 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 tab to the Service DEFAULTS field and enter the name of one of your service classes, followed by PF3. If you do not associate one of your service classes with each of the classifications, it will default to a SYSTEM level service class, which is an automatically provided high-level service class.
- ▶ When you have assigned all the classifications, PF3 to the main menu and select the Utility option at the top of the screen. You can validate your new definitions using one of the options. (The validate function appears to check syntax, but makes no attempt to do more than that.)

---

<sup>26</sup> As already noted, there is a huge body of documentation about detailed WLM definitions. You can search this material for information about goals, durations, and other groupings.

- ▶ Use PF3 to leave the WLM dialog. You should receive a panel offering to save your definition to a dataset. Use this option to save to the PDS you defined for it. At this point you could also use option 3 to discard all your work! This can be useful if you are experimenting with the WLM dialog panels.
- ▶ There are many other choices in the WLM dialog panels, but the options we described are sufficient to create a basic WLM service definition.

If you followed these steps you now have your service definition 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 already have the name you 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; this is likely to include ADCD.Z24A.WLM or a similar name.

- ▶ Select one of the saved names; you will probably select your own dataset or the ADCD dataset if you want to revert to the default AD-CD WLM service definition.
- ▶ You should then see the main WLM dialog panel. Select the Utilities option in the top line of the screen, and select option 1 to Install the service definition. You will then see a subpanel asking permission to overwrite whatever service definition is currently installed. Reply YES to this question.
- ▶ If you use the **D WLM** operator command you will 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 presented. This will activate the service policy and a **D WLM** command should confirm this.

Is there any point to building your own WLM service definition when working with an AD-CD system on 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 and this is most meaningful if the system has a sustained workload approaching 100% CPU utilization. Adjustments are made at something like 10 second intervals. A zPDT system with a few TSO users editing, compiling, and unit testing modules presents an erratic workload that does not respond to WLM very well. However, a heavy test workload that involves batch, DB2, and perhaps extensive (simulated) terminal interaction might respond to WLM very well.

## 12.20 TELNET

z/OS AD-CD releases typically allow you to connect to TCP/IP port 1023 with a simple **telnet** command connection. The old **telnet** command might no longer be available, but a common equivalent is the **putty** command used in telnet mode. (As is well known, any form of **telnet** is very insecure, but there might arise special needs 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 in the appropriate place:

```
telnet 23/tcp
otelnet 1023/tcp
```

Current z/OS AD-CD 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 member(s) in your PARMLIB.

## 12.21 Delete logstreams

The default location for log streams (for the AD-CD system) is the xxSYS1 volume . These streams can sometimes grow to fill the volume. A typical 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 should be done only if the log stream file is not being managed; that is, it is not represented in current couple data sets.

## 12.22 SMF

Some of the recent AD-CD z/OS releases produce excessive SMF output. This is especially true if type 99 records are included. In these cases the default parameters were as follows:

```
SYS(NOTYPE(14:19,62:69),EXITS(IEFU83,IEFU84,IEFACTRT,IEFUSI,IEFUJI,
IEFU29),NOINTERVAL,NODETAIL)
```

If you have absolutely no interest in SMF data, we suggest you 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) and perhaps RMF data (types 70 - 79) we suggest the following line to produce minimal recording:

```
SYS(TYPE(30,70:79),EXITS(IEFU83,IEFU84,IEFACTRT,IEFUSI,IEFUJI,
IEFU29),NOINTERVAL,NODETAIL)
```

Recent AD-CD systems process full SMF datasets automatically via a supplied IEFU29 exit, discarding the output. If you want to process SMF data yourself, remove the IEFU29 exit that appears twice in the default SMFPRMxx parameters. (We suggest that you eliminate recording SMF record types that you do not want before doing this.)

The uses of the various exits specified in the SMF parameters, if present, is beyond the scope of this book. If you have access to Cheryl Watson's Tuning Letters, the 2009 letters contain a useful and practical introduction to SMF usage.<sup>27</sup>

If you use logstreams for SMF and want to determine the quantity of each SMF type being collected, you might use the following job:

```
//OGDEN101 JOB 1,OGDEN,MSGCLASS=X
// EXEC PGM=IFASMF DL,REGION=0M
//OUT DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  LSNAME(IFASMF.GENERAL,OPTIONS(DUMP))
  OUTDD(OUT,TYPE(0:255))
/*
```

---

<sup>27</sup> For more information, see <http://www.watsonwalker.com>.

This job assumes your LOGSTREAM for SMF is named IFASMF.GENERAL, which is the name used in the parallel sysplex system for zPDT described in other documents. Your name might differ.

If you want to list a few details about the status of the LOGSTREAM, you might 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)
/*
```

## 12.23 Disabled waits

You might sometimes see a message in your zPDT Linux window such as this example:

```
Warning! Disabled Wait CPU 0
$ d psw                                <--you can issue this 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 PSW. The last three characters of the PSW should contain a wait state code; an extended code may be also present in other characters of the PSW. The following list provides abbreviated information about standard wait state codes. See the System Codes manual for more complete information about each code. Note that some wait states are restartable; again, see the System Codes manual for more information.

```
Partial list (see the notes at the end of the list)
002 - During IPL, an I/O operation was not initiated (1)
003 - IPL cannot continue; subchannel is not operational for IPL or IODF device (1)
004 - During initialization, I/O not initiated (1)
005 - I/O interruption during IPL and unit check (2)
006 - I/O error during IPL processing; SYSRES or IODF volume (2)
007 - During initialization console not available (3)
009 - System build error. (z/OS problem; should be rare)
00A - Cannot find SYS1.LINKLIB or SYS1.CSSLIB in catalog (4)
00B - Master scheduler abended (4)
00D - Master scheduler abended (4)
00E - Problem on SYSRES volume (SYS1.NUCLEUS) (2)
00F - IPL volume does not contain IPL text (5)
013 - Error during NIP (6)
014 - Recursive program checks (6)
017 - Unit check during IPL (2,5)
019 - IPL program in error (6). Also seen if IPLing an older operating system (that does
not support CZAM) on a z14 operating in the normal CZAM mode.
01B - SLIP requests wait (7)
01C - Recursiveabend in FRR (8)
020 - Reconfiguration initialization failed (9)
022 - Page fault - devices quiesced or not ready (10)
023 - Trace initialization failed (11)
025 - Duplicate entry point in nucleus (6)
02E - ASM detected too many I/O errors (10)
030 - ABEND during NIP (6)
031 - No UCB for SYSRES (1, 2)
032 - NIP module missing (6)
```

033 - I/O error in BLDL during NIP (6)  
 035 - Could not find entry point in nucleus (6)  
 037 - DSCB for SVCLIB, PARMLIB, LINKLIB could not be read (12)  
 038 - Not enough main storage (11)  
 039 - DASD mount conflict (1,13)  
 03A - Error building LPA (6)  
 03B - Required module is not in LPA (6, 14)  
 03C - ASM found not enough paging storage (15)  
 03D - Error building page tables (11)  
 03E - Not enough page slots to back master scheduler initialization (15)  
 03F - NIP function invoked incorrectly (6)  
 040 - ABEND during NIP (6)  
 044 - Machine check during NIP (6) (Try IPLing again, at least once!)  
 045 - NIP could not initialize RTM (6)  
 046 - Program check during NIP (6)  
 04A - TOD clock in error (16)  
 050 - Alternate CPU recovery (ACR) entered recursively (16, 6)  
 051 - ACR had error (software) (6)  
 052 - ACR error (hardware) (16)  
 053 - PC or PC/AUTH failed (17)  
 054 - Error in member loaded into nucleus (6,14)  
 055 - IPL cannot find necessary member in SYS1.NUCLEUS (14)  
 056 - NIP error (6)  
 059 - Unidentified return code for BLDL during NIP (6)  
 05A - ACR tried to remove last CP (16)  
 05C - NIP cannot find catalog pointer in nucleus (18,12)  
 05D - During NIP, could not find DSCB for catalog (12)  
 05E - Error reading master catalog (2)  
 060 - ASM detected errors in page tables (6)  
 061 - TOD clock errors during STCK instruction (16)  
 062 - Channel path error (16)  
 063 - NIP storage problem. SQA too small? (4)  
 064 - NIP error and RTM not initialized (6)  
 065 - NIP issued type 3 or 4 SVC before they were available (6)  
 06F - I/O problem - unusual (1,2,10,13)  
 070 - NIP: insufficient contiguous main storage (6,11)  
 071 - System or operator initialized a restart  
 072 - No more workspace for IPL (6,11)  
 073 - IPL program waiting for I/O or external interrupt (16)  
 074 - IPL program contains a logic error (6)  
 075 - IPL program could not load a module (4,14,6)  
 076 - IPL found non-fullword relocatable address constant (6)  
 077 - SVC entry point cannot be resolved (6)  
 078 - Master catalog could not be opened (2,4,6)  
 07B - Required processor facility not available (19)  
 07C - Initialization error, configuration problem. (18,1,see Codes manual)  
 07D - IEASYSxx PARMLIB member is bad; in error (4)  
 07E - Unable to obtain LSQA storage for SVC (11,6)  
 081 - SYS1.NUCLEUS occupies more than one extent  
 082 - System joining sysplex needs maintenance  
 083 - Incorrect address in PSA (16)  
 084 - RTM error (17,6)  
 085 - ASM warm start problem (1,10,20)  
 087 - System removed from sysplex (normal situation)  
 088 - IPL: error in LOADxx or NUCLSTxx (18,4,6)  
 089 - NIP found an error in a UCB (6)  
 08A - WTO error going to wait state  
 08C - WLM has recurring error (6)  
 08E - SYSEVENT error (6)  
 08F - Failure rebuilding work queues (6)

09x - SPINLOOP problem (See Codes manual for more information)  
 0A1 - Excessive SPINLOOP unresolved (21)  
 0A2 - XCF encountered cross system problem  
 0A3 - Unable to join global GRS  
 0A4 - ETR problem  
 0A5 - HCD problem (remember: zPDT does not support dynamic reconfiguration)  
 0A7 - Insufficient ESQA or ECSA storage (11)  
 0B0 - Could not recognize IODF specified in LOADxx (18,1)  
 0B1 - LOADxx member problem (18,1)  
 0B2 - No devices in IODF (18, or you created a bad IODF)  
 0B3 - Incorrect information in IPL parameter  
 0B4 - UIM specified unidentified device number (22)  
 0E0 - SIGNAL failed during NIP (16)  
 0E1 - SIGP STOP failed because processor was not operational (16)  
 0E3 - Insufficient virtual storage to initialize CSA (4)  
 0E8 - During NIP, the machine check handler failed (6)  
 101 - Program in supervisor state requested too much SQA (6)  
 102 - Program in supervisor state requested more pages of SQA than available (6)  
 104 - While proccessing ABEND SVC, program check occurred recursively (6)  
 110 - System detected hot I/O from device other than DASD (16; note MVS console messages)  
 111 - System detected hot I/O on DASD device (16, note MVS console messages)  
 112 - System detected hot I/O on reserved DASD (16, note MVS console messages)  
 113 - Failure during channel path recovery (16)  
 114 - Previous error affected SMP operation. See manual.  
 115 - DASD containing paging dataset is unavailable. (10)  
 116 - During restart, detected missing interrupt for paging device (16)  
 11A - Error during SVC 26 (6)  
 201 - Failure while creating COMMTASK (6)  
 202 - During system initialization, creation of console communications failed (6)  
 204 - Error during allocation (6)  
 205 - Attempted to load a module that was not in LINKST (6)  
 206 - Sysplex initialization operator message prompt  
 5C7 - Error during processor or system termination (6)  
 A00 - DAT error for system address space (16,6)  
 A01 - Error on only online processor (16)  
 A18 - Unsolicited Device end on paging volume; AVR failed. (complex)  
 A19 - Can no longer perform I/O (16)  
 A1E - Time-of-day clock failed (16)  
 A1F - Processor controller not available; TOD sync cannot occur (16)  
 A20 - System found page in FLPA that is not fixed (6)  
 A21 - Segment table entry for MLPA, PLPA, FLPA, or xFLPA is incorrect (6)  
 A22 - Error (probably hot I/O) invoked disabled console communication facility (see manual)  
 A23 - Program check during machine check handling on only online processor (16)  
 A24 - Loop while running machine check handler on only online processor (6,16)  
 A26 - Machine check on only online processor; interruption code incorrect (16)  
 A27 - Problems during machine check interruption handling  
 A28 - DAT-off machine check handler cannot start DAT-on machine check handler (6,16)  
 A29 - Problems stopping processor after program or machine check (6)  
 A2A - System detected LPA page that is not on paging data set (6)  
 A2B - Error in extended storage (16)  
 A70 - Console unavailable during NIP (3)  
 A71 - Reconfiguration problem (9; see codes manual)  
 A7A - Service processor interface failed (16,9)  
 B01-B1D Wait states used by the 3203/3211 utility  
 B20-B24 Wait states used by the stand-alone IOCP program  
 CCC - Wait state generated by QUISECE command  
 D0D - SMF had resource shortage (too much SMF output requested? Memory too small?)  
 E02 - Should never happen (16)

EC7 - Severe error in Unix System Services (6)  
FFx - Non-IBM program created a wait state

The suggested actions provided by the following notes assume you are using z/OS (probably the AD-CD system) in a normal manner. That is, we have these assumptions:

- ▶ You have not modified the system.
- ▶ You are not working with authorized programs or code.
- ▶ You have not installed middleware that operates as authorized code.
- ▶ You are not actively disrupting the hosting Linux environment for zPDT.

Review these notes:

1. Check that your devmap contains the necessary volumes and that they are at addresses supported by the IODF of your z/OS system. For example, with an AD-CD z/OS system have your 3390 volumes addresses in the range A80 -AEF. Restart zPDT and try again. If necessary, verify your emulated 3390 volumes (using an `alckkd xxxx -rs` command.)
2. Verify that your IPL parameters point to the correct volumes for SYSRES and the IODF volume. With the typical AD system these are addresses A80 and A82. Possibly these volumes are corrupted. Verify your emulated 3390 volumes (using an `alckkd xxxx -rs` command), restart zPDT and try to IPL again.
3. z/OS wants a NIP console. This is address 700 in existing AD-CD z/OS systems. Be certain a 3270 session is connected to this address (or whatever NIP console address is specified for *your* z/OS system). Also be certain the Linux window you used to start zPDT is open, as this could provide an alternative, limited NIP console function in some cases. If a 3270 session at address 700 is available, try to IPL again.
4. Have you altered a working system? Changed key PROCLIB members? Changed the catalog line in the LOADxx member in SYS1IPLPARM? This is not recoverable. You must IPL another z/OS system (or restore volumes for this z/OS.)
5. Ensure you IPL the correct volume. Verify your devmap and try again.
6. Internal z/OS problem. Probably not your fault unless the volume is corrupted. Try doing an IPL of z/OS again. You might need to IPL another z/OS system or restore volumes for this z/OS system.
7. Someone set a SLIP trap. This is probably a user-caused wait and whoever set the SLIP trap should know how to proceed. System can be restarted.
8. Typically a system error, but might be an error in a software product. Try doing an IPL of z/OS again, possibly without starting recently installed middleware.
9. You attempted a reconfiguration option that is not available for zPDT.
10. Possible devmap or PC disk error or emulated 3390 corruption error. Stop zPDT and verify 3390 emulated volume formats with `alckkd xxxx -rs` commands.
11. Be sure you defined enough memory for your z System. Try increasing the memory size in your devmap and restart zPDT. (Consult zPDT documentation to understand the maximum z System memory definition recommended for your configuration.) Most zPDT users run z/OS with *at least* 4000 MB z System memory defined.
12. Did you IPL the correct volume? Has someone deleted data sets on this volume? Has someone deleted system data sets?
13. Check your devmap carefully. Are two 3390 definitions pointing to the same file? Verify that the 3390 devmap entries point to the correct Linux files. Fix your devmap and restart zPDT.
14. Be sure no one deleted members in any system libraries. This is probably not recoverable. You need to IPL another z/OS system and possibly repair this z/OS system.

15. You may have started a large program (such as WAS) and you do not have enough space in your paging data sets. You might add paging data sets.
16. Restart zPDT. If the error persists, contact your zPDT provider.
17. Probably due to bugs in a software product. Were you starting a new product when this happened? This is not recoverable. Re-IPL and try again.
18. Verify whether anyone modified SYS1.IPLPARM and whether you are using a new LOADxx member in this library. If so, the new member has an incorrect catalog line. IPL with a “standard” load parameter (to use a “standard” LOADxx member).
19. See the zPDT documentation for information about what facilities and functions are available through zPDT. If you are unable to resolve the problem, contact your zPDT provider.
20. Try a cold start (CLPA).
21. Try restarting zPDT with fewer processors defined in your devmap. Contact your zPDT provider if the problem persists with fewer devmap processors defined than you have real processors in your PC.
22. Check your devmap. Devmap and IODF must have compatible device addresses. Are you attempting to use an unsupported device?





## Additional zPDT notes

This chapter contains various topics, in no particular order. This information is not required for basic zPDT operation but it is helpful for better understanding of zPDT and for more advanced uses.

### 13.1 Known z/VM issue

If a zPDT base PC is running z/VM and if the PC is placed in a hibernate state (by closing the laptop lid, for example) for a lengthy period then z/VM *might* fail with a PRG009 ABEND. In the example we have the “lengthy period” is 17 hours, but this is variable. The underlying problem relates to TOD clock skews in the PC and there are multiple variables involved. This situation is very rare and, at the time of writing, is a permanent restriction for zPDT.

### 13.2 Free zIIPs”

Starting with 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 zPDT instance and toward the requirement for the number of PC cores to be at least equal to the number of emulated processors. Also, there may not be more zIIPs than CPs.

For example, a user with a 1090-L02 token (two licenses), using a PC with four cores, could configure two CPs and one zIIP, if our recommendations about having at least one more core than z processors.

A user with a 1090-L03 token (three licenses), using a PC with four cores, and with three CPs, has no spare cores for a zIIP. This leads to discussion about PC Hyper-Threading.

## 13.3 PC Hyper-Threading

zPDT recommends that PC Hyper-Threading be disabled in the BIOS of any PC running zPDT. We sometimes encounter z/OS “excessive spinloop” messages when running with Hyper-Threading enabled, and disabling Hyper-Threading usually eliminates these messages. While we can not 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 such spinloop messages recently while running fairly simple batch and TSO sessions. However, this does not mean that the spinloop situation does not exist; it only means that any spinloop conflict does not last long enough to trigger a z/OS message.

Why is this topic relevant? Consider a user with a 1090-L03 token (three licenses) and a PC with four cores, a common configuration. If this user wants to verify that his applications use zIIP processing he has choices:

1. Ignore our recommendation to have at least one more core than zPDT processors. He can simply add a zIIP to his devmap:

```
processors 4 cp cp cp ziiip      #this requires three zPDT licenses
```

The disadvantage is that general zPDT performance can be substantially degraded.

2. Reduce the number of CPs. This retains our recommendation of more cores than zPDT processors, but possibly reduces throughput due to the smaller number of CPs:

```
processors 3 cp cp ziiip        #this requires two zPDT licenses
```

In a sense, this “wastes” a zPDT license.

3. Experiment with Hyper-Threading. With Hyper-Threading enabled (in the BIOS) the PC appears to have eight cores. The user might configure this:

```
processors 5 cp cp cp ziiip ziiip #this requires three licenses
```

This violates the zPDT recommendation against Hyper-Threading but might be useful in an experimental situation.

Based on limited experience in this environment (Hyper-Threading enabled, 1090-L03, three CPs, one zIIP) we noted the following:

- ▶ The zIIP was used, as expected, by our simple Java applications. This was important in verifying that our applications were behaving as expected.
- ▶ When the zIIP was not in use (but with the zIIP defined, and with Hyper-Threading enabled) our sample workload used slightly more CPU time<sup>1</sup> and slightly less elapsed time than when there was no zIIP and Hyper-Threading was disabled. That is, there was a small CPU “cost” to having Hyper-Threading enabled, while there was an elapsed time savings.
- ▶ Our use of Java was light and we did not have any Db2 work. Heavier workloads in these two areas might change our conclusions.

*All formal zPDT testing by IBM occurs with Hyper-Threading disabled.* This is the “safe” way to run zPDT. If you have a larger server, with many cores, there is no need to consider Hyper-Threading. If you have a very common environment, with an L03 token and a four-core PC, and verifying zIIP processing is important to you, you might consider our experience related here.

---

<sup>1</sup> The results mention here are from SMF type 30 records.

**Important:** We have not experimented with the C/C++ compiler SMP option. This option provides for parallelization of a program and this might create more exposure for spinloop problems when using Hyper-Threading.

We have also seen problems involving very heavy z/OS Java loads (such as zOSMF) when using Hyper-Threading.

## 13.4 cpuopt statement

The `cpuopt` statement specifies optional parameters for the CPs. These are the only valid parameters at the time of writing:

<code>cpuopt asn_lx_reuse=on</code>	<i>(no blanks in operand)</i>
<code>cpuopt asn_lx_reuse=off</code>	<i>(no blanks in operand)</i>
<code>cpuopt zVM_CouplingFacility</code>	<i>(no blanks in operand)</i>
<code>cpuopt alr=on,zVM_Coupling</code>	<i>(abbreviations)</i>
<code>cpuopt ZARCH_ONLY=NO</code>	<i>(upper case, no spaces)</i>

The `asn_lx_reuse` operand may be abbreviated as `alr`. The `zVM_CouplingFacility` operand may 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, defaults to YES, and is the standard mode of operation for IBM z14 systems. It causes the z14 to IPL in normal zArchitecture mode. Specifying `ZARCH_ONLY=NO`<sup>2</sup> causes zPDT to IPL in ESA390 mode, but otherwise operate as a z14 system. This is a non-standard configuration that *might* be useful with older operating systems that are not designed to IPL in zArchitecture mode. Use of this option creates an environment that is not supported by IBM, and should be used with caution. (The `ZARCH_ONLY` function is sometimes documented as the CZAM facility.) IPLing an older z/OS operating system (that does not support IPL in zArchitecture mode) on a 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” and this is the normal mode of operation for zPDT. This mode matches the relevant architecture of IBM z10 and later machines. When this parameter is set to “off” zPDT indicates that the LX and ASN REUSE facility is not present. This mode *might* be useful for running early z/OS releases. The use of `alr=off` produces an environment that is not supported or tested by IBM. While it may be useful for working with earlier z/OS releases, the user must assume all responsibility for correctness of operation and the correctness of results.

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.

## 13.5 Copying DASD volumes

*Never* use Linux to copy an emulated DASD volume while zPDT is active and potentially using that volume; that is, if the emulated volume is in the devmap. For example, *never* use

<sup>2</sup> The `ZARCH_ONLY=NO` options also disables the CM390 feature of z14 architecture. This should have no direct effect for most users.

Linux **cp** or **rsync** or similar commands to copy a potentially active volume. The Linux commands are unaware of any buffer or cache information 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 zPDT is not active, any Linux commands that copy files can be used to copy (“back up”) an emulated volume. An emulated DASD volume is a single Linux file. As a practical matter, the **gzip** command is often used to copy and compress a volume.

## 13.6 Read-only and shared DASD

Emulated DASD volumes may be used as read-only volumes. This is done by setting the Linux permissions to disallow writing to the Linux file that contains the emulated volume. For example, assuming we have a 3390 volume stored in /z/WORK02, the following Linux command<sup>3</sup> makes it read-only for the owner (normally the Linux userid that started zPDT), the owning group, and all other user IDs:

```
$ chmod 444 /z/WORK02      (you may use other forms of chmod, of course)
```

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 zPDT.

### Older z/OS levels

When z/OS accesses the volume an error message might be displayed (probably due to an attempt to update VTOC statistics) but operation continues in a read-only mode.<sup>4</sup> You can browse data sets, for example. If you attempt to change a data set (with ISPF edit, for example) an error message is produced and you must cancel the SAVE operation. The error messages are similar to the following example:

```
(on the MVS console): IOS000I 0AA0,01,WRI,1D.....  
(MVS console and TS0): IEC212I 414-04,IFG0201.....
```

We informally used basic sequential datasets, PDS datasets, and PDS/E datasets without problems. We were unable to use VSAM datasets on a read-only volume, there might be techniques to bypass this restriction. z/OS APAR OA50068 provides z/OS support for read-only volumes and provides extended documentation for their use. zPDT implementation of read-only (at the Linux permissions level) differs from the technique described in the APAR and this might produce operational differences.

### Recent z/OS levels

z/OS has recently added more formal support for read-only volumes. This is documented in the *Using Datasets* manual (SC23-6855). With this support, JCL support and a degree of VSAM support are available for read-only volumes.

### 13.6.1 Shared read-only volumes

You can share read-only DASD volumes. The sharing is done at the Linux level and is 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.

<sup>3</sup> Depending on your Linux file ownership you might need to operate as *root* to issue this command.

<sup>4</sup> We noticed that z/OS 2.1 is less likely to produce error messages than earlier z/OS releases.

**Important:** The usage described here requires all access to the volume to be read-only. It is not suitable, for example, for allowing one zPDT system to have write access, while all other sharing systems are read-only.

There are many ways you might configure such operation. We implemented a simple configuration as follows:

1. We placed the intended read-only 3390 volumes (that is, the Linux files containing these volumes) in a separate directory on our first Linux system. In our case, we named this directory /z3. We changed the permissions on each file in the directory to read-only. (This implies, of course, that the data sets on these volumes already contain whatever source code and data we want to share among our z/OS systems.)
2. We started an NFS server on this Linux system. In our case, we used the openSUSE YAST interface to configure the NFS server functions and indicated 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 have various methods of configuring NFS server operation and you must use whatever interface is appropriate for your Linux.
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 new mount point. In our case, we named this /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. Note that we needed 192.168.1.80:/ and not 192.168.1.80:/z3<sup>5</sup> in the **mount** command. In principle, we should then see the shared volumes by issuing **ls /z3** command on the client machine. In practice, we sometimes needed to reboot 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 then start zPDT and IPL 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 probably means ending zPDT operation and possibly closing down Linux. (We did not explore the exact details of this scenario.) You end zPDT on the server, change the permissions for the volumes you want to alter, start zPDT, IPL z/OS, make the changes, end zPDT, and change the file permissions back to read-only. At that point, you can reboot the client system (or systems) and ensure that they can “see” the volumes at the mount point. This is not a convenient scenario and read-only operation is not appropriate for volumes that are frequently updated.

We noticed that zPDT startup on the client was a little slower than with previous operation and z/OS shutdown (with a **quiesce** command) was definitely slower.

<sup>5</sup> This does not appear completely logical to the author, but **mount -t nfs 192.168.1.80:/z3 /z3** did not work.

## 13.7 zPDT log files

zPDT writes a number of log files based in the zPDT user's home directory. If Linux user *ibmsys1* starts 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 very interesting otherwise. The log types include the following:

<code>awsstart_pnnnn_000001.log</code>	<i>(one line; created when starting zPDT)</i>
<code>CPU_QUEUE_SEQMEWNT...gz</code>	<i>(IBM internal use only)</i>
<code>dv_OA8A_pnnnn_000001.log</code>	<i>(created when starting each device)</i>
<code>emily_uim_pnnnn_000001.log</code>	<i>(created when starting UIM server)</i>
<code>log_console-pnnnn-t2018-11-03_10.12.00.txt</code>	<i>(console log)</i>
<code>summaryFile</code>	<i>(IBM debugging only)</i>
<code>tnport12_pnnnn.log</code>	<i>(created by aws3274)</i>
<code>uimclient_pnnnn_000001.log</code>	<i>(created by UIM client startup)</i>

The “pnnnn” in these descriptions is the Linux process ID (pid) that was used. Older log files are not immediately cleared and will typically have various pid numbers.

One of the log types might be more interesting to zPDT users: these are the *log\_console* files. There may be several of these. The name includes a time stamp indicating when the log file was started. In the example above, the *log\_console* was started November 3, 2018 at 10:12 AM. The *log\_console* file includes the following:

- ▶ Startup messages from zPDT.
- ▶ Asynchronous messages from zPDT that are displayed in the Linux command terminal used to start zPDT.
- ▶ zPDT commands entered in any Linux terminal window.<sup>6</sup>
- ▶ Messages from z/OS (or other operating system) that are directed to the “HMC hardware console.” These messages are displayed in the Linux command terminal used to start zPDT.

Notice the two categories of messages that are displayed in the Linux command terminal used to start zPDT. If this Linux command terminal (“window”) is closed, these messages are not *displayed* anywhere but are still *written* to the *log\_console*.

The following technique might be useful to zPDT users:

(Linux command):	<b>oprmsg 'V CN(*),ACTIVATE'</b>	
(Linux command):	<b>oprmsg d a,1</b>	<i>(a z/OS command)</i>

This activates the “HMC hardware console” as a z/OS operator terminal. The **oprmsg** commands may be entered in any Linux command window, but the resulting messages from z/OS are shown only in the Linux window that was used to start zPDT and are also placed in the *log\_console* file. If you wish to automate (from Linux) various z/OS commands and responses the technique described here might be useful.

The optional *oprmsg\_file* statement in the zPDT devmap is used to direct placement of the *log\_console* file to a different location. For example:

```
[system]
memory 20G
processors 3
oprmsg_file /tmp/sysmessages
...
```

<sup>6</sup> To be more precise, any Linux terminal window operating under the Linux userid that started zPDT. This detail is relevant if multiple zPDT instances are operational.

causes the log to be written to /tmp/sysmessages instead of a file in the /home/ibmsys1/z1090/logs/ directory.

Numerous logs can accumulate in the default *logs* directory. You can force clearing by using the `--clean` option with the `awsstart` command.

## 13.8 Very large PC memory

Large PC memory usually improves zPDT performance. Large memory helps avoid paging and provides a large Linux disk cache function. In very rare cases, very large PC memory<sup>7</sup> 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 very active large tables is not being used.
- ▶ The z/OS system is quickly producing large numbers of multiple asynchronous disk operations (especially *write* operations).
- ▶ RAID5 usage might be involved. RAID5<sup>8</sup> requires two disk writes for each logical write operation.
- ▶ PC memory is large enough that the Linux disk cache can absorb many gigabytes of disk output without actually writing to disk.
- ▶ At some point Linux decides that it has too many “dirty” pages in the cache and must write them to disk before allowing more disk operations.<sup>9</sup>
- ▶ z/OS starts seeing disk timeouts, typically as MIH (Missing Interrupt Handler) actions, and may create a variety of error messages. z/OS will usually eventually resolve the situation correctly.

We have seen this situation when using pathological test cases. We have very rarely seen it in normal usage. One extreme example we investigated was as follows:

- ▶ The PC server had 160 GB memory.
- ▶ z/OS was running under z/VM.
- ▶ RAID5 was being used.
- ▶ The user had defined five new, full 3390-9 volumes for JES2 spool space.
- ▶ When cold starting JES2 the first time, it started formatting the five volumes, asynchronously, in parallel. (JES2 formatting is very fast and efficient, contributing 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 correctly in this situation.

In this particular case, running the same z/OS functions without using z/VM allowed the JES2 formatting to complete correctly. (There were still MIH and other disk error messages but the functions completed correctly.) Subsequent use of the z/OS system under z/VM operated normally.

Recent Linux levels appear to better handle situations such as described here. We stress that the overrun situations described here are very, very rare. We often use PCs with large memory (192GB, most recently) with no problems during “normal” z/OS operation.

<sup>7</sup> In this case, “large” means *at least* 100 GB.

<sup>8</sup> Most RAID adapters have sizable cache memory. This discussion assumes volumes of data such that the RAID adapter cache is relatively small.

<sup>9</sup> This is probably an inaccurate description of exactly how Linux manages the disk cache, but it describes the general situation.

## 13.9 Token dates and times

The zPDT tokens (and equivalent software-only license manager) remember the latest date and time that they obtain from the underlying Linux system. The token must never see the date or time move backward. If this happens, a *time cheat* message is produced and zPDT does not start.

For example, if you set the PC date ahead several months (perhaps to test an expiration function in the application you are developing) and you use 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 zPDT supplier.

If you *must* temporarily change the PC clock (when not using zPDT), remove the token before doing this and reset the clock to the current time before connecting the token again. If you move a token among multiple PCs, you should take care that the hardware time-of-day clocks reflect times close to each other on all the machines.<sup>10</sup>

The **settod** command provided with zPDT provides a way to test software by using different dates; this does not change the value of the PC hardware clock or the Linux software clock.

## 13.10 Typing OPRMSG too many times

The default z System console (normally on an HMC) is emulated by zPDT by using the Linux window that was used to start zPDT. Operating system output sent to the default console appears in the Linux window. To enter z System commands from the default console (that is, from the Linux window) use the **oprmsg** command, as in this example:

```
$ oprmsg 'CN(*),ACTIVATE'
```

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 plus character to create oprmsg)
$ + 'cn(*),activate'
$ + d a,l
```

A space is needed after the plus sign (+), just as a space is needed after the **oprmsg** command before the command text is entered. Remember that single quotation marks might be needed to prevent the Linux shell from processing special characters such as parentheses.

## 13.11 Important Linux command window

The Linux command window that is used to enter the **awsstart** command is important. Asynchronous messages from zPDT are sent to this window. (User commands sent to zPDT can be entered from any Linux window that is operating under the same userid that started zPDT). Asynchronous messages include zPDT error messages (such as an unexpected CTC disconnection) and z/OS messages directed to the HMC console.

---

<sup>10</sup> This is a technical statement. Your zPDT license agreement may restrict this usage.

If you inadvertently close the Linux command window that is used to start zPDT, you will not see these asynchronous messages. At present, there is no way to recover the functionality of this window.

## 13.12 Linux “out of memory”

In rare circumstances we see an “out of memory” error message from Linux, typically when starting zPDT. Assuming your zPDT runs correctly most of the time, we believe this message is usually related to fragmentation of the shared memory locations in Linux. zPDT uses Linux shared memory extensively and 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 zPDT is not running), but the easiest solution is to reboot Linux. We emphasize that this is a very rare condition.

## 13.13 The crontab and sudo entries

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 UIM functions result in additional 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 userid you authorize. Do not remove these entries from */etc/sudo*.

## 13.14 Dynamic configuration changes

zPDT does not support dynamic changes to an operational devmap. The selected devmap is read when zPDT is started (by using the **awsstart** command) and any changes made to the devmap after that point are not effective unless zPDT is stopped and restarted.

The most common reason for wanting to change the devmap is to alter the DASD configuration, usually by adding additional volumes. This can be done with a little planning. The basic requirement is to include “spare” DASD devices in your devmap, using device numbers (addresses) valid for your IODF. Here is an example:

```
[manager]
name awsckd ABCD
device 0A80 3390 3990 /z/H1RES1      (normal emulated DASD definitions)
..... (other DASD definitions)
device 0AB0 3390 3990                (spare device definition)
device 0AB1 3390 3990                (spare device definition)
```

We could use a spare device to add a new work volume, as follows:

```
$ alcckd /z/LOCAL4 -s500 -d3390      (create new emulated volume, 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. We must run an ICKDSF job, as follows:

```
//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 **VARY AB0,ONLINE** command and the volume is ready for use. You probably will want to change your devmap so that the volume is routinely available when zPDT is started the next time.

**Intense Linux I/O:** The **alckd** command, when creating a new emulation volume, creates intense Linux I/O while it is formatting the volume. This could disrupt or slow concurrent z/OS operation, especially if significant z/OS jobs are running.

## 13.15 Security exposures

While most zPDT usage occurs in environments where base Linux security is not a significant concern, the following items might be of concern to some users.

### 13.15.1 Reducing root usage

After zPDT is installed, few functions normally require running as *root*. The most common are the commands involved in updating the token licenses and the **clientconfig** command. This use of *root* can be avoided by taking the following steps:

1. Select a userid (not *root*) of someone who will be allowed to use the token update commands. These commands are **SecureUpdateUtility** (for zPDT releases prior to GA5), **Z1090\_token\_update**, and **Z1091\_token\_update**.
2. As *root* (probably when installing zPDT) issue the command:

```
# SecureUpdate_authority -a <userid>          (specify your selected userid)
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; it makes an entry in the */etc/sudo* file. To remove a userid from the authorized list issue:

```
# SecureUpdate_authority -d <userid>
or
# /usr/z1090/bin/SecureUpdate_authority -d <userid>
```

3. Thereafter, use the **sudo zpdtSecureUpdate** command<sup>11</sup> while operating as the selected userid. This command automatically switches to the */usr/z1090/bin* directory and executes the appropriate commands with *root* authority. The **sudo** function provides temporary *root* authority and the **zpdtSecureUpdate** command temporarily switches to the required directory, executes the appropriate token administration command, and returns to the previous directory.

<sup>11</sup> This applies to zPDT release GA5 and earlier releases.

4. Select a userid (not *root*) who will be allowed to use the **clientconfig** command and issue the following command as *root*:

```
# clientconfig_authority -a <userid>
```

A userid may be removed from the authorized list by using a **-d** flag instead of **-a**.

5. Thereafter the indicated userid can use the **clientconfig** command.

As a practical matter, the same userid may 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.

### 13.15.2 Linux suid use

The zPDT system software operates as a normal Linux application with a few exception. The eDMosa module (that provides the emulated OSA function) operates with Linux *root* privileges. That is, it uses suid permission to operate as *root*, and the permissions are “world” executable. While we have no indication that this has happened, it might be possible for a non-zPDT Linux user to execute eDMosa and, in some way, use this to compromise the base Linux system. The suid module is also visible to programs that scan Linux for “unapproved” suid files.

You can remove the “world” executable permission, as follows:

1. Select (or create) a Linux group for use only by zPDT functions. The installation instructions in “zPDT installation” on page 95 suggest creating a group named *zpd*t, although the specific name is not important. You can use the GUI administrative functions of your Linux to add the group (and associate selected userids with the group).
2. Change the ownership of eDMosa to this group. For example,  

```
# chgrp zpd /usr/z1090/bin/eDMosa
```
3. Change the permissions for eDMosa,  

```
# chmod 4750 /usr/z1090/bin/eDMosa
```
4. Remember that any Linux userid that is to be used to start zPDT must be a member of the new group. Other userids should not be members of this group.

### 13.15.3 Gen1 token server monitoring

The Gen1 token software used with zPDT has a web monitoring function. This is not relevant to normal zPDT operation, but 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
```

## 13.16 z1090instcheck

The **z1090instcheck** command should be run after the 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 shown here are examples. The shmmax value should be at least as large as stated in the note. The shmall value 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 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 recommended for heavy OSA usage, or larger frames.
- ▶ Lines 9, 10, and 11 reflect parameters for core image files. These are potentially important if 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 distributed with 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. This 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 have caused zPDT problems in the past.

Additional checks might be added to later versions of **z1090instcheck**. Note that some checks are absolute while others look for values in a range thought to be appropriate. Your output from **z1090instcheck** may differ slightly from what is shown here as minor details may change with zPDT updates or new Linux distributions.

## 13.17 zPDT build information

The `/usr/z1090/bin/librarybuild` text file contains information about the levels of Linux used to create the current copy of zPDT. In general, you should not use a Linux that is earlier than the libraries noted in this file.

## 13.18 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 use 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 must be done when zPDT is not active.<sup>12</sup>
- ▶ zPDT is started, an IPL of an operating system is done, and the volumes are used normally.
- ▶ Later (when 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 use might be for a demonstration or benchmark. After the demonstration or benchmark is completed, the volumes can be restored to their original state.

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 has been 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:

\$ <b>alcckd</b> /z/WORK01 -ve	<i>enable versioning for the indicated volume</i>
\$ <b>alcckd</b> /z/WORK01 -vr	<i>restore volume to original content</i>
\$ <b>alcckd</b> /z/WORK01 -vc	<i>commit the changes to the volume</i>
\$ <b>alcckd</b> /z/WORK01 -vi	<i>inquire about the versioning status of the volume</i>
\$ <b>alcckd</b> /z/WORK01 -vd	<i>disable versioning (if no changes have been made)</i>

These examples use an emulated volume stored in `/z/WORK01`. You would specify the name of the Linux file containing your volume, of course.

---

<sup>12</sup> zPDT can be active if the selected volumes are not in the active devmap.

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; subsequent changes to the track simply 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 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 additional “original tracks” are stored. In an extreme case, where every track on the volume is changed, the Linux file will grow to twice its original size. In typical cases, relatively few tracks on a volume are changed through normal use and the Linux file growth is minor.

If versioning was enabled for a volume, but there have been no changes to the volume, the **-vd** option can be used to disable versioning. If there have been changes to the volume (causing the versioning function to start operation) then the **-vd** option is rejected. At this point 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 have been versioned.

## 13.19 zPDT messages

Most messages issued by zPDT have unique message numbers. The **msgInfo** command can be used to obtain more information about a message. An example of its use might be as follows:

```
$ awsckmap aprof1 (check my 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 needed but
AWSINF013I     if errors are present the DEVMAP cannot be used to start system.
```

All message numbers are in the form **AWScnnnns**, where:

- ccc is the component code issuing the message.
- nnn is the message number within the component.
- s is the message severity (**D**ebug, **I**nformation, **W**arning, **E**rror, **S**evere, **T**erminal)

The message code 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 named **Z1090\_MSG** to control message formatting. It may be set to **FULL** (the default), **CODE** (which will only print the message number and no text), **TEXT** (which prints

the message text and no code) and SHORT (which drops the AWS prefix on the message number).

## 13.20 TCP/UDP ports

Several TCP/IP ports are used by normal 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 use of these ports. Port information is as follows:

- ▶ The zPDT token device is accessed through TCP/UDP and requires a port number. A *well known* port number has been assigned for this device; this is Linux port 9450. The Unique Identity Manager (UIM) function uses Linux port 9451; this is only meaningful when using a remote license manager.
- ▶ The Gen2 license server, if used, uses port 1947.
- ▶ If you use the awsctc device manager, be aware that 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, “DASD volume migration” on page 289) uses port 3990 by default.
- ▶ The STP function uses port number 4567 by default.
- ▶ The Safenet Gen1 license manager provides a status web display page using port 7002.
- ▶ If you are operating your base Linux system through VNC, the default port(s) are 5900+N, where N is the VNC display number.

## 13.21 Remote operation

A zPDT system (including z/OS) may be operated remotely, using a Linux command window (Telnet, VNC, or SSH) and TN3270e sessions connected to the base Linux on the zPDT system. No special techniques or setups are required. As with local operation, having the TN3270e session for the z/OS console connected 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 command windows used to control zPDT.

Complete remote operation of a Linux system running zPDT might be most convenient using VNC. When doing this, it is most convenient to start zPDT from a VNC window. This allows the owner to disconnect the VNC session and reconnect to it later without affecting 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.

## 13.22 Many zPDT devices

The current version of zPDT has a maximum of 2048 emulated I/O devices for an instance. If you define more than approximately 100 devices you may need to change Linux kernel definitions. You may need to alter:

- ▶ The total amount of Linux shared virtual memory. This is the *kernel.shmall* parameter.
- ▶ The amount of Linux shared virtual memory available to a process. This is the *kernel.shmmax* parameter.
- ▶ The maximum number of shared memory segments. This is the *kernel.shmmni* parameter.

- The parameter that relates to the number of Linux semaphores available. This is the *kernel.msgmni* parameter.
- The numbers of semaphore devices available. These are the *kernel.sem* parameters.

The default values for these parameters appears to vary considerably with various Linux releases. In many cases the default *shmall* and *shmmax* parameters are very large and need not be changed. The other parameters *might* need to be changed.

Each I/O device defined in your devmap requires approximately 300 KB of shared virtual memory. The total of these, plus the memory defined for your z System, must be within the allowed total Linux shared virtual memory. Your defined z System memory size (plus perhaps 20% overhead) must fit within the shared virtual memory available for a process.

The zPDT developers do not have exact formulas for these parameters. The default numbers created by the `/usr/z1090/bin/aws_sysctl` script are reasonable for typical zPDT systems with a few dozen defined devices and, perhaps, around 10 GB defined for z system memory.

We found the following formula for *msgmni* to be reasonable:

```
kernel.msgmni = (350 + 3 * number-of-I/O-devices)
```

This is not an exact formula, but it should produce safe values. Also, if you have more than approximately 128 emulated I/O devices you should use `ulimit -m` and `-v` statements mentioned in Chapter 5, “zPDT installation” on page 95.

Using multiple zPDT instances, each with a large number of devices, may exhaust the semaphore space in Linux, resulting in a hang when starting a zPDT instance. This 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 zPDT instances), generally in excess of a total of more than about 1200 devices, may require the *shmmni* value to be increased:

```
kernel.shmmni 8192              (default is typically 4096)
```

In one rather large case, using 2048 devices in a single zPDT instance, we defined the following:

```
kernel.shmall=4300000000
kernel.shmmax=10000000000      (make larger for larger z System memory)
kernel.shmmni=32000
kernel.msgmni=6500
```

These numbers might not be optimum, but they worked for us.

## 13.23 Startup scripts

We created several trivial startup scripts for our z/OS zPDT system, such as this example:

```
$ gedit start00
cd /home/ibmsys1
awsstart aprof1
sleep 6
echo 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
```

We could then start our system with a single `./start00` command. Our scripts (differing only in the IPL parm data) were trivial and could be enhanced in many ways. Be certain to allow sufficient time for the 3270 sessions to start before executing the IPL command.

A better alternative to a Linux startup script is to embed Linux commands in the zPDT device map. The method for doing this is explained in 3.2, “System stanza” on page 34.

## 13.24 Suspend and Hibernation

Laptop PCs, in conjunction with Linux software, offer suspend (save to RAM) and hibernation (save to disk) functions to effectively save the current PC state and transition to a non-operational state that uses very little power. ***zPDT is not tested for, and does not support these functions.*** These functions become problematical when timed token drivers or time-dependent LAN connections are involved, such as with zPDT license servers.

Some zPDT users have reported successfully using these functions. One technique is to use a z/OS **quiesce** command before the suspend operation. This allows z/OS to complete in-progress I/O and prepare other internal operations for a restartable disabled wait. After restarting the PC, a zPDT **restart** command (entered in the same zPDT operations window that was visible before the hibernation) should cause z/OS to resume operation.

We have had erratic experience with this on different PCs and different Linux versions. In one case, using Fn-7 after “opening the lid” restored the graphic screen and the **restart** command worked as expected. In another case we were never able to restart the graphic screen, while in yet another case the graphic desktop restarted with by itself.

Some users have 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.<sup>13</sup> Linux commands such as the following might help:

```
# service shk_usb restart                (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 verb could be **start** or **restart**.

You must determine if hibernation or suspend works for you; *remember that it is a Linux function, and not part of zPDT.*

<sup>13</sup> After these error messages zPDT might acquire the missing license automatically. If this happens, z/OS is probably unaffected and continues to run. The error messages are seen only in the Linux command window used to start zPDT.

## 13.25 Channel connections

zPDT does not support direct channel (Parallel, FICON or ESCON) connections to traditional z System devices. Many zPDT customers obtain their zPDT systems through the Information Technology Company (commonly known as ITC). ITC offers a method of indirectly connecting zPDT<sup>14</sup> to FICON or ESCON DASD control units. This method is based on a hardware device, sold by ITC, known as a zData Appliance and is shown in Figure 13-1. The general connection arrangement is shown in Figure 13-2. (The trademark “zData Appliance” is owned by Information Technology Company, LLC.)



Figure 13-1 Typical zBox

A zData Appliance can use FICON or ESCON connections (up to two channels) and can connect to the Linux zPDT system via Ethernet or USB. The diagram shows only FICON and Ethernet because these are the most common configurations. The diagram contains a simplified version of a z Systems disk unit. In practice, a configuration often contains FICON switches (or ESCON directors) and many more channels connecting to various z System processors.

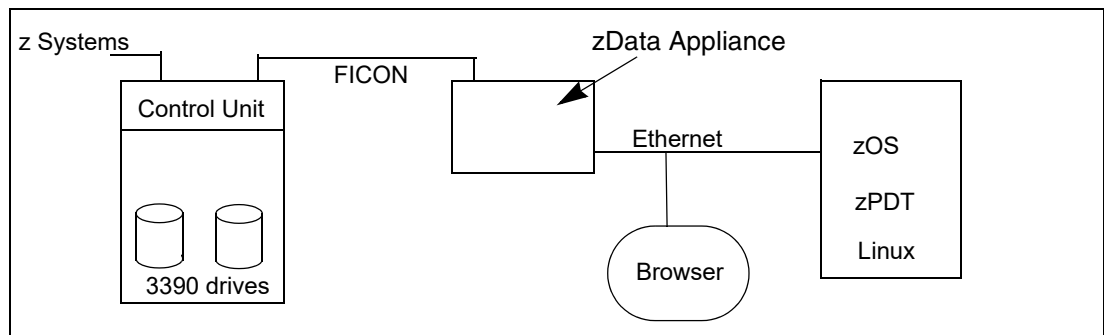


Figure 13-2 zBox connection concepts

The zData Appliance contains considerable internal storage, equivalent to multiple 3390 volumes. The general concept is that the zData Appliance, when commanded to do so, copies a complete 3390 volume image from the z System control unit to internal appliance storage and zPDT then accesses it via an NFS mount. The zData Appliance can be directed

<sup>14</sup> ITC names their complete systems “uPDT” but we use the more familiar name zPDT in this description. More information is available at [www.itconline.com](http://www.itconline.com) or [sales@itconline.com](mailto:sales@itconline.com).

to rewrite the 3390 image (possibly updated by programs running under zPDT) to its original location. Configuration and control of the zData Appliance is through a browser session connected to it. It is important to understand that the 3390 drive(s) are not *shared* in the general sense of z Systems shared DASD.

The user (on zPDT and the zData Appliance) must coordinate 3390 access with the users of the z System involved. There is no serialization mechanism involved. When the zData Appliance is instructed (via the browser) to read or write a 3390 volume, it does it regardless of concurrent activity on the z System. The necessary coordination should generally ensure there are no open datasets on the volume when a copy (in either direction) is taking place, and establish plans to manage catalogs such that the current catalog(s) are available for any VSAM datasets involved.

This is a brief description of the product; more information is available from ITC.<sup>15</sup>

## 13.26 Channel measurement blocks

With zPDT GA9, selected fields in channel measurement blocks are meaningful for zPDT users, as indicated in Figure 13-3. As specified in the z architecture, the unit of timing is 128 microseconds.

Format 0		Format 1	
*SSCH count	*sample count	* SSCH count	
* device connect time		* sample count	
* function pending time		* device connect time	
device disconnect time		* function pending time	
CU queuing time		device disconnect time	
device active only time		CU queuing time	
device busy time		device active only time	
* initial cmd response time		device busy time	
* Data meaningful with zPDT		* initial cmd response time	
		interrupt delay time	
		I/O priority delay time	

Figure 13-3 Channel Command Blocks

Equivalent fields in a subchannel extended measurement word have a granularity of .5 microseconds. Please note that the data in the measurement blocks (and measurement word) might not match that which would be found on a large IBM z system. Various Linux cache and zPDT buffer operations (not visible to z/OS) can result in measurement times that appear impossibly short for a standard IBM z system.

<sup>15</sup> Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 or sales@itconline.com.

## 13.27 x3270 scripting

The x3270 emulator has a scripting function that is not very 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 onto TSO. This script<sup>16</sup> is entered into a simple Linux file in your home directory and this file is later executed.<sup>17</sup>

```
#!/bin/bash
x3270 -socket &                                (start the x3270 session)
pid=$!                                         (no internal spaces here)
until x3270if -p $pid '""> /dev/null         (four single quotes 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("xxxxxxx")'           (the TSO password)
x3270if -p $pid 'enter'
x3270if -p $pid 'CloseScript()'               (see following note)
```

The CloseScript functions leaves the x3270 session running and you may manually enter whatever TSO commands are desired. The **wait** function blocks 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 screen output, but it does not provide a way to wait for a particular output string. A user could save the current 3270 screen (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 documented for x3270 scripting is a **MoveCursor(row,col)** function. It is important to realize that the row and col parameters in this function are zero-based, while the row and column numbers displayed on the bottom of an x3270 screen are one-based.

## 13.28 Scenarios for awstape

The practical use of the zPDT awstape device manager can be a bit confusing to those without hands-on experience with mainframe tape units. In particular, the physical mounting of a tape volume on a tape drive must be emulated, as well as the reading, writing, and positioning of the “tape” involved. The zPDT **awsmount** command emulates the physical mounting of a tape volume and the zPDT **ready** command emulates a tape drive (with a mounted tape) going from *not-ready* to *ready* state. Both these commands produce an emulated interrupt that informs the z System operating system of the action.

<sup>16</sup> We thank Paul Mattes, the primary author of x3270, for help in bypassing a timing problem while producing this script.

<sup>17</sup> Remember to use a **chmod** command to make the Linux file executable.

There is only one awstape format. The z 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 created (containing the emulated tape volume) is the same in all cases. For example, you can create an awstape volume using an emulated 3420 drive and later read using an emulated 3490 drive.

The following scenarios describe several approaches to using awstape.

### 13.28.1 Create a new labelled tape volume

Consider the following job that calls for a standard label tape volume having the volume serial TAPE99. Furthermore, assume that we do not already 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 command window you now 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. At this point the job waits until the operator mounts the requested tape. The zPDT user must enter an **awsmount** command in a Linux command window. This satisfies the z/OS mount request. In the example here, we specified Linux file name TAPE99.aws.<sup>18</sup> This Linux file name is completely arbitrary; it need not contain the tape volser or an *aws* suffix. We suggest using a naming convention, but this is up to the user. In this example, file TAPE99.aws is created in the Linux directory that was used to start zPDT. Alternatively, a complete Linux path name could 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 causes the z/OS messages shown, and allows the z/OS operator to enter the desired volume serial number. Once this 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 logically mounted on the drive an **-o** flag specifies that whatever emulated tape file is currently mounted is to be replaced by the newly specified file. There are many more **awsmount** options described in “The **awsmount** command” on page 59.

<sup>18</sup> In this example the Linux file does not yet exist. It is created by the **awsmount** command.

## 13.28.2 Quick SL volume creation

The **aws\_tapeInit** command can be used to create a standard label awstape volume, with no data and with minimal label information. An example is:

```
$ aws_tapeInit tape01 /tmp/mywork/TAPE01
```

This creates a Linux file (/tmp/mywork/TAPE01) in awstape format, with the standard label TAPE01. (The conversion of the volume serial to upper case EBCDIC is automatic.) The new volume is simply a Linux file; this command performs no *mount* function.

## 13.28.3 Use an existing labelled tape

If we 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 command window 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 desired awstape volume.

## 13.28.4 Automatically premount existing labelled tape

We can use the zPDT awscmd device manager (described in “The awscmd command” on page 207) 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 (via the **awsmount** command) that z/OS has not yet requested. This invokes the z/OS automatic volume recognition (AVR) function, making the volume available for a subsequent request for the volume. This avoids operator interaction.

- The example uses in-stream SYSIN to provide the **awsmount** command. A more general approach would point to a dataset 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 used by the awscmd function. This is described in the chapter on awscmd.

We found this technique to be rather fragile, with the setup and JCL requiring some experimentation to make it work. However, this approach might be considered by zPDT users working toward a simplistic automatic tape library function

### 13.28.5 Use special CCWs to manipulate tape volume

zPDT GA9 provides two new channel commands for working with awstape. These functions have been requested by a number of zPDT customers. The active devmap must have the **--MNTCCWS** option specified on the awstape *name* statement to enable this function.

Channel command 4B is used to change the Linux file associated with an emulated tape unit. Channel command 54 is used to obtain the Linux file name currently associated with an emulated tape unit. These operations are not reflected back to the z operating system in any way; they *do not* create device interrupts; they cannot satisfy a z/OS operating system *mount* request.

These 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 unlabelled tape. Other z operating systems might have more permissive tape drive usage and could 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 STAPE0        Assume BLP on DD statement
        TM STAPE0+48,X'10'  Did OPEN work?
        BZ ERROR1          If not,branch
* Write name of target awstape file (name coded as a constant in this example)
D        MVC SECB,=F'0'     Zero my 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          Execute the CCW
        WAIT ECB=SECB       Wait for it
D2       TM SECB,X'20'      Completed OK?
        BZ ERROR2          If not, branch
* Read the current Linux file name that is mounted (to verify mount)
```

```

C      MVC   SECB,=F'0'      Zero my ECB
      LA     1,SCCRNAME      @(CCW)
      ST     1,SIOB+16       Place address in IOB
C1     EXCP   SIOB            Execute the CCW
      WAIT   ECB=SECB        Wait for it
C2     TM     SECB,X'20'      Completed OK?
      BZ     ERROR3          If not, branch
CLOSE  CLOSE  (STAPE0)
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, work areas
*
SAVEAREA DC    24F'0'
STAPE0   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(STAPE0)       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 the above job might not be obvious. Details include the following:

- ▶ z/OS requires that a device (an emulated tape drive in this case) must be “open” before it can be accessed.<sup>19</sup>
- ▶ The TAPE DD statement in the first job step specifies a tape volume (UNLABB). When the job is run the zPDT user must use the zPDT **awsmount** command to mount an unlabeled awstape volume to satisfy the z/OS mount request. The tape volume that is mounted this

<sup>19</sup> There are techniques for working around this requirement, but these techniques require authorized programs working with z/OS interfaces not intended for external use.

way is not actually used; 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) then uses the CCWs to change the Linux file name that contains the desired awstape volume. In this simplistic example, the new Linux file name is coded in the program (/z/zos23/UNLABC) while a more realistic approach would probably have the name in another dataset.
- ▶ The mount CCW (4B) is followed by the query CCW (54). The query will, we assume, read the same name that was mounted. However, the query must be coded and it cannot be command chained to the mount CCW. Failure to follow the mount with the query results in an ABEND 214-08 when the CLOSE is issued.<sup>20</sup>
- ▶ 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 unlabelled tape (UNLABB) that was specified in the first step and passed to the second step via the RETAIN option. However, the first step changed the actual file to be used (/z/zos23/UNLABC).
- ▶ Remember that this mount CCW (4B) does not perform a z/OS mount; that is, there is no mount interrupt generated and z/OS is unaware that the awstape volume name has been changed “under the covers.”
- ▶ 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 desired Linux file name. The length field in the CCW must be correct and must not include trailing blanks with the Linux file name. The length field for the query command in the example is arbitrary. You would probably have a buffer longer than needed and use the residual count from the IOB to determine the actual length of the returned data. In the example, the query data is not examined.
- ▶ The general flow described here is unlikely to work for labelled tapes in z/OS/

As mentioned above, these CCWs have been requested by several zPDT users. Some experimentation might be needed to effectively use them and this is left to the user.

### 13.28.6 Premounted tape

It might be convenient to have an emulated tape mounted and ready to use, without MVS console interactions, just after z/OS is IPLed. There are several ways to approach this depending on your exact requirements. As a starting point, let us assume you have a standard labelled tape volume (an “awstape” volume) in /home/ibmsys1/TAPE01. We assume the tape (with tape label) was previously created. In these examples we assume your z/OS IODF defines address 561 as a 3480 device.<sup>21</sup> There is nothing unique to 3480 drives in these examples; we use it in order to present concrete definitions and commands.

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 zPDT and IPLing z/OS you can run one job using this volume without encountering any z/OS mount messages. The job would contain a DD statement something like the following:

<sup>20</sup> This was true at the time of writing and the zPDT developers are not clear about why z/OS creates this ABEND.

<sup>21</sup> This is consistent with z/OS AD-CD releases.

```
//SYSUT2 DD DISP=(OLD,KEEP),VOL=SER=TAPE01,UNIT=561,DSN=MY.DATA
```

The DISP could be also be NEW. After the first z/OS job using this tape completes, the tape is logically dismounted and additional uses will produce z/OS console MOUNT messages. In general, you would respond to these messages, via a Linux command window, with

```
$ awsmount 561 -m /home/ibmsys1/TAPE01
```

2. Instead of including the tape file name in the *device* statement, you could 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 is IPLed,<sup>22</sup> but before the z/OS job is submitted, you could enter the following via a Linux command window:

```
$ awsmount 561 -m /home/ibmsys1/TAPE01
```

This, in effect, provides an unsolicited mount and triggers the AVR (Automatic Volume Recognition) function in z/OS. The next z/OS job that calls for the tape volume will immediately run without producing any MVS console MOUNT messages.

4. You can issue a **mount** command from the MVS console:

```
MOUNT 561,VOL=(SL,TAPE01)
```

This produces a mount message on the MVS console. You would then enter the awsmount command via a Linux window:

```
$ awsmount 561 -m /home/ibmsys1/TAPE01
```

This usage differs from the other examples in that the tape volume remains mounted after jobs using it end. That is, 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-labelled tape.)

5. Many z/OS systems invoke the VTAMAPPL program during system startup. This program executes a series of MVS commands (a “script”). The MOUNT command can be included in this script. For example, recent z/OS AD-CD system have a script in PARMLIB member VTAM00 that could be modified similar to the following:

```
S RRS,SUB=MSTR
S TSO
...
...
PAUSE 2
VARY 561,ONLINE
MOUNT 561,VOL=(SL,TAPE01)
```

---

<sup>22</sup> Assuming you have tape drive 561 in you devmap, as in the previous example.

```
PAUSE 2
VARY 561,ONLINE
```

This example assumes you have 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 may be necessary to complete the mount. In some cases, for reasons the author does not understand, the mount remains pending after the **vary 561,online**. In this case, a **ready 561** command from a Linux console completes the mount.

### 13.28.7 awstape utilities

(The following text was moved from the chapter about SCSI tape drives.)

zPDT users often build a substantial “tape” library on disk, all in awstape format. The **tapeCheck** command can be used to verify that a file (which corresponds to a tape volume) is in the correct awstape format.

```
$ tapeCheck /z/TAPE01                (verify format of awstape file)
```

The **tape2file** command copies an awstape file to a simple byte stream in a Linux file, removing 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, using the same conversion conventions as the awsrdr device manager:

```
$ card2tape /tmp/myLinux.stuff /z/tape01    (copy without translation)
$ card2tape -a /tmp/myLinux.xyz /z/tape01   (force translate 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 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 uncompress 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.



## Tape drives and tapes

Starting with zPDT GA10, general SCSI tape drives are no longer supported. Only Fibre-connected 3590 drives are supported. However, the awsscsi device manager from earlier 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 zPDT

### 14.1 zPDT 359x Tape Support

ITC<sup>1</sup> has worked with zPDT development by testing changes to the awsscsi device manager that are designed to more fully support FCP attached IBM 3590/3592 tape units. Although these devices could be physically attached to a zPDT system prior to these awsscsi enhancements, they functioned only in a rather basic mode, emulating a 3490 device. The updated support allows these units to operate in full 359x mode, with minor exceptions.<sup>2</sup>

All tests were run using openSUSE 12.2 and the Xfce user interface/desktop. The base zPDT system was driver 45.18 and the AWSSCSI modifications were applied on top of that release. Other versions or flavors of Linux were not tested and results may be unpredictable.

#### 14.1.1 The FCP Adapters

ITC has 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 offered with IBM System x, such as the 8Mb Emulex, IBM part number 42D0485. Although most testing was with Emulex adapters because they seemed more available, we did test with at least one Qlogic adapter (Dual port, 8 Mb IBM part number 42D0510). In all cases, the FCP driver included in openSUSE 12.2 functioned correctly and newer/different drivers were unnecessary.<sup>3</sup> 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 tested were PCIe format adapters.

<sup>1</sup> Information Technology Company LLC, 7389 Lee Highway, Falls Church, VA 22042 or sales@itconline.com.

<sup>2</sup> One exception is that automated tape library functions are not supported.

<sup>3</sup> This may not be true for other/older Linux distributions or for FCP adapters that are not so widely used.

## 14.1.2 3590/3592 Tape drives

Testing was performed with 3592-J1A and 3592-E05 tape units, and the 3590-H11 with 10-cartridge autoloader.<sup>4</sup> There are no configuration options for 3590 devices. These drives are also known as TS1120 models.

All 3592 drives need to be configured<sup>5</sup> for RACK installation; not for LIBRARY installation. Each drive has two ports. The port you intend to use must be configured as PORT SPEED=Auto Negotiate, TOPOLOGY=L-Port; HARD ADDRESS is suggested to be set to x88 although not required.

Performance is limited by the x86 architecture of the underlying hardware platform and the configuration of the Linux operating system. Consequently, performance numbers cannot be expressed in meaningful terms. Experience has shown that, generally, 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 devmap in order to match a z/OS IODF entry, as in this example,

```
device 590 3590 3590 /dev/sg3
```

## 14.2 Discontinued support

**Important:** The remainder of this chapter describes SCSI usage that is untested and without formal support, starting with zPDT GA10. This text is included for customers who might still be using SCSI tape drives. This text will probably be removed in the next edition of this book.

SCSI tape drives may be used in two ways:

- ▶ The awsscsi device manager allows SCSI tape drives to be used by z System programs.
- ▶ Several Linux utilities that directly use SCSI tape drives are provided with zPDT. These utilities may be used when zPDT is not active. These utilities are not associated with devmaps or a device manager.

However, not all SCSI tape drives may be usable. The usability depends on the exact tape drive model, the firmware level, the firmware options selected, and the exact SCSI adapter (and firmware level) that is used. Several different tape drives have been tested, but we cannot guarantee that *your* tape drive will work. We *strongly* suggest that you work with your zPDT supplier to understand your SCSI tape drive environment.

This chapter discusses three categories of SCSI tape drives:

- ▶ Tape drives using traditional SCSI cables. These are emulated as 3420, 3480, or 3490 drives, regardless of the actual nature of the tape drive.
- ▶ IBM 359x tape drives connected with fiber cables that are defined as 3420, 3480, or 3490 drives in the devmap. These are subject to the same comments as when using parallel SCSI cables.

<sup>4</sup> ITC also has available single and dual drive models (E05, E06, and E07) that are complete ready-to-run units with power supplies, IBM FICON adapter, FICON cable, CE-Panel, and warranty.

<sup>5</sup> Configuration changes are entered through an optional CE panel. The drives obtained by the zPDT developers did not require configuration changes. You might discuss the need for a CE panel with your 359x provider.

- IBM 359x tape drives connected with fiber cables that are defined as 3590 drives in the devmap.

## 14.2.1 The awsscsi device manager

Selected SCSI tape drives may be used as “real” tape drives during zPDT operation. The awsscsi device manager is used and allows the SCSI tape drive to appear as a 3420, 3480, 3490 or 3590<sup>6</sup> 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 arbitrary CUNUMBR operand. This example defines a tape drive at address (device number) 581. If z/OS is being used, then the z/OS IODF must have a corresponding device (IBM 3490) defined for this address. (z/VM detects devices dynamically and would find a 3490 at this address.)

The *appearance* to software (as a 3490 in the example above) has no direct relation to the actual SCSI device type. In this case, /dev/sg5 might be a DLT drive, for example, that has no physical characteristics of a 3490 drive.<sup>7</sup>

As shown in Figure 14-1, Linux SCSI devices may be addressed through two interfaces. The relationship between the st and the sg interfaces can be complicated because the sequence number used for a given drive is typically not the same number.

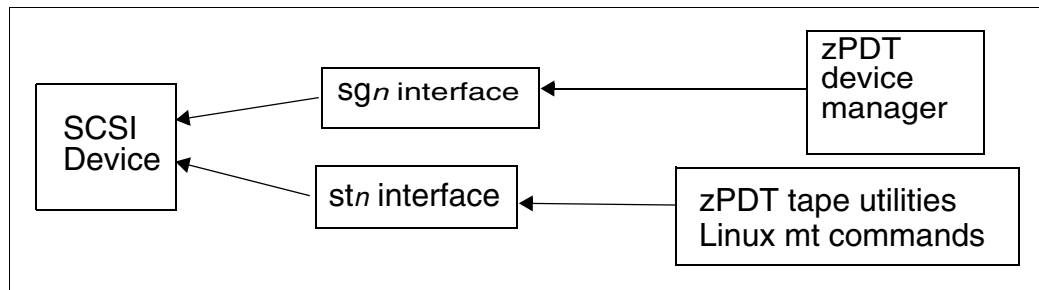


Figure 14-1 SCSI driver interfaces

The Linux device for the SCSI tape drive can be changed with the **awsmount** command, as in this example:

```
$ awsmount 580 -u /dev/sg5          (disassociate sg5)
$ awsmount 580 -m /dev/sg3          (mount different drive)
```

The last operand of the device statement (/dev/sg5 in the example) denotes the SCSI device to be 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
```

<sup>6</sup> Usage as a 3590 drive requires a fibre adapter and a “real” 359x drive.

<sup>7</sup> IBM did not formally test DLT drives.

The `/dev/stN` and `/dev/nstN` interfaces are for *sequential tape devices*. The first tape drive on a Linux system would be `/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.<sup>8</sup> The `/dev/stN` and `/dev/nstN` interfaces are used with the 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.<sup>9</sup> Unfortunately, the N value for a given 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).<sup>10</sup> The first (and only) tape drive on a Linux system would be `/dev/st0` but it might be `/dev/sg7`, for example.

## Determine 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 you specify for the **awsscsi** device manager, but any Linux utility uses `/dev/st0` or `/dev/nst0` for the same tape drive. The devices listed in `/proc/scsi/scsi` are in `sgN` order; `sg0` is first, `sg1` is second, `sg2` is third, and so forth.

You can list both the `st` and `sg` interface numbers for all SCSI tape drives 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
```

Always remember that the `sg` number can change from one Linux boot to another if there is any configuration change.

## Permissions

There is another issue with both `/dev/stN` and `/dev/sgN` devices: the permission bits must allow usage by zPDT. Some Linux systems allow general user access to `/dev/stN` devices by default; 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*
```

<sup>8</sup> The `/dev/stN` device automatically rewinds (whatever this might mean for the actual device) when the device is closed. The `/dev/nstN` devices do not provide an automatic rewind.

<sup>9</sup> 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.

<sup>10</sup> The `sg` numbers are assigned in ascending order by bus and by device on the bus. However, since Linux treats a number of non-SCSI devices as though they were SCSI, these bus and device numbers are difficult to predict in advance.

```

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 zPDT) can read and write to it. This change (by using **chmod**) is lost when Linux is rebooted, but is suitable for many situations. Always verify the correct sgN number first, by listing /proc/scsi/scsi.

A more permanent change can be made by modifying Linux boot functions. Unfortunately, there are two problems with this:

- ▶ The method of modifying Linux boot functions differs with different Linux distributions. For example, we might change /etc/rc.local, or /etc/rc.d/rc.local, or /etc/init.d/boot.local, or another file, depending on the exact Linux distribution.
- ▶ We can easily make a general change for stN and nstN devices, but we cannot easily make a general change for sgN devices. We should *not* change the permissions to allow universal access to all /dev/sgN devices; this would allow easy destruction of our Linux system.

We could place the following lines in /etc/init.d/boot.local (assuming our particular Linux 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 particular SCSI device

```

Do not use **chmod 666 /dev/sg[0-9]** because this allows anyone to directly access all the SCSI devices in your system. Unless you always have exactly the same devices powered up when you boot Linux, you cannot safely predict the sgN number of a given device.

## Block counts

There has been confusion about zPDT use of SCSI-attached tape drives defined as 3490 devices. This definition means 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; the largest count it can hold is approximately 4 million. If you write to a SCSI-attached tape drive defined as a 3490 (under zPDT) you will receive an end-of-media indication after writing approximately 4 million blocks. At this point z/OS performs EOVS functions and calls for a new tape cartridge (depending on your JCL, of course). 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 written by another system (that was not emulating a 3490 drive) the cartridge might contain more than 4 million blocks. This should work correctly provided the zPDT z System program does not read the block count. If it does, and if the tape position is past the 4 million block point, the z Systems program will indicate a block count error. What happens at that point depends on the design of the particular application program. A tape cartridge with more than approximately 4 million blocks is not fully compatible with 3490 emulation.

## 14.2.2 Parallel SCSI adapters

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

- ▶ 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 have informally<sup>11</sup> 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 distributions, with Linux kernels below the level used in openSUSE 11.2, did not work with these SCSI adapters on some of our systems. This condition is likely to change with future Linux distributions. If parallel SCSI operation is important to you, we *strongly suggest* that you discuss your zPDT configuration with your zPDT provider.

- ▶ There is no *defined* IBM support for these configurations.
- ▶ Parallel SCSI adapters, cables, and devices can be complex. There are different data path widths, single-ended and differential circuits, low-voltage and high-voltage versions, and a variety of terminators. If you are not familiar with this area, we *strongly suggest* you obtain expert help in configuring your system.
- ▶ The newest SCSI devices use fiber connections instead of parallel (wire) connections.

## 14.2.3 Specific hardware tested

**Tip:** Be aware that 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 involved in SCSI tape drive usage can be complex. We strongly suggest that you talk with a knowledgeable zPDT provider about your SCSI tape drive requirements.

IBM testing 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 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 was at firmware level 5BG4 as determined by the itdtinst1.2LinuxX86 tool from IBM.

These drives were tested with IBM 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 was at Emulex LightPulse x86 BIOS Version 1.71A0, firmware ZS2.50A, as displayed during the BIOS startup.

<sup>11</sup> This statement was for zPDT releases prior to GA10.

The Linux device drivers (provided in the Linux distribution) were determined by using the command `dmesg | grep Emulex`:

- RHEL 5.3 (64-bits): Emulex LightPluse Fibre Channel SCSI driver 8.2.0.33.3p.
  - openSUSE 11.1 (64-bits): Emulex LightPluse Fibre Channel SCSI driver 8.2.8.7.
  - SLES 11 (64-bits): Emulex LightPluse Fibre Channel SCSI driver 8.2.8.14.
- Q Logic Corporation ISP2432-based 4Gb Fibre Channel to PCI Express HBA (rev 03). This 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-bits): Qlogic Fibre Channel HBA Driver 8.02.00.06.05.03-k.
  - openSUSE 11.1 (64-bits): 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 this with these commands:

```
$ su                                (switch to root; may 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
```

Notice that we use the `/dev/stN` devices rather than the `/dev/sgN` devices for these commands. Also, note the Linux must have the `mt` command installed.

With our older drives and adapters the Linux drivers for Emulex and Qlogic both defaulted to a maximum block size of 32K. Block sizes larger than 32 K are typically not used by application programs; however, large block sizes may be used by backup programs (such as ADRDSSU for z/OS) or by virtual tape managers. (The newer drivers used with the IBM 359x drives defaulted to 64K maximum block size.)

The following Linux commands were used to set `def_reserved_size` to 65536:

```
$ su                                (change 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 64k)
# 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 64KB reserved buffer size for data transfers. Note that these commands do not make a permanent change. A Linux reboot puts the default size back to 32768.

Recent Linux releases may have changed these interfaces.

## 14.2.4 zPDT SCSI utilities

Two zPDT utilities can work directly with SCSI tape drives, assuming the Linux system has access to the SCSI adapter. Standard awstape-format files (in Linux) can be moved to and from SCSI tape devices 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 awstape file)
$ scsi2tape -c /dev/st0 /z/mytape2    (compress the awstape file)
```

These two commands are typically used when zPDT is not active.<sup>12</sup>

## 14.2.5 Linux SCSI tape utilities

Linux can provide basic tape utility functions through the **mt** package. This package is not required for zPDT, but may 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.

## 14.2.6 Practical advice

Most SCSI tape drives do not use vacuum columns to help manage tape start/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 this:

- ▶ If the program (including the operating system elements) issues tape read or write commands quickly enough, the tape drive will run the tape at full speed.
- ▶ If the program (including the operating system elements) does not issue reads or writes quickly enough, the tape drive will stop after the current data block. The next read or write might cause the tape drive to “backhitch” (that is, back up the tape for a distance) and then restart forward movement. This is done to ensure the tape is “up to speed” for the next read or write operation.

This backhitch movement can greatly reduce the effective data speed of the drive. Not all drives encounter this; 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 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 could then process the emulated tape volume copy instead of the real SCSI tape volume. This might result in much faster processing of your tape data.

The SCSI adapters 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 zPDT.

---

<sup>12</sup> If the SCSI devices are not in the active devmap, these utilities can be safely used while zPDT is active.



## DASD volume migration

The zPDT package includes a client-server utility for moving 3390 volumes from a remote z/OS or z/VM system to zPDT. The server portion of this utility runs under z/OS or z/VM on the remote z System. The remote is typically a large z System, but it could be another zPDT system. The client portion runs on the base Linux of your zPDT machine.<sup>1</sup> The client and server are connected via TCP/IP. This utility is especially useful when transferring many volumes to a 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 used by zPDT and writes it as a Linux file. You then use this file as an emulated volume under zPDT.

There is no “reverse migration” function available through this client/server operation. If you need to copy a 3390 volume from zPDT to a “real” 3390 you can use the ADRDSSU program, as described in 12.11, “Moving 3390 volumes” on page 226.

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. It 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 may take some time.

A conceptual overview is shown in Figure 15-1 on page 290.

---

<sup>1</sup> zPDT need not be operational while this utility is being used. Due to expected LAN and disk activity, it is probably better to use the migration utility when zPDT is not active.

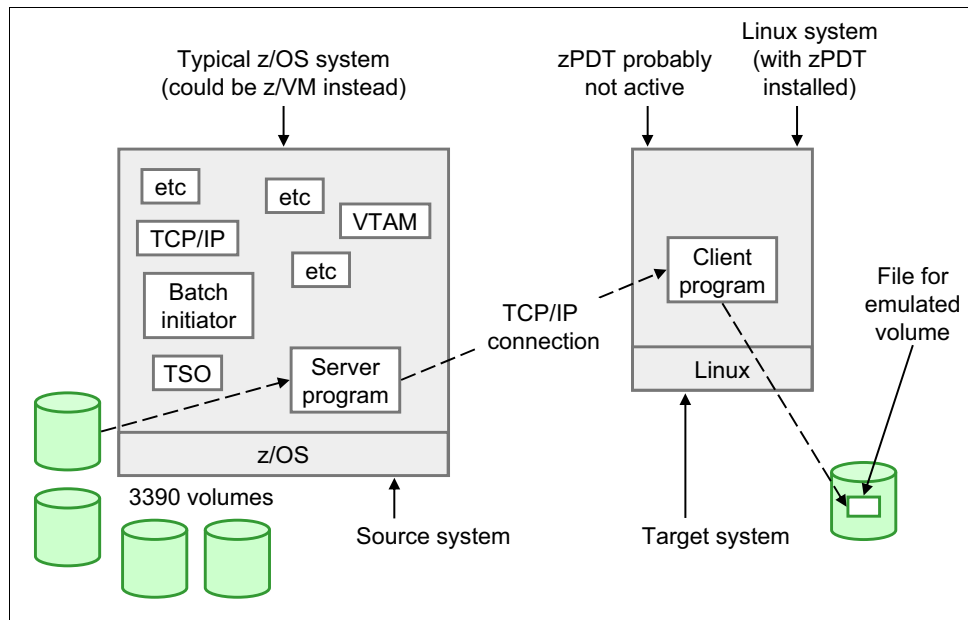


Figure 15-1 Volume migration overview (z/OS version)

## 15.1 Warnings

There are limitations on simply 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 zPDT or to the migration utility described in this chapter. A z/OS disk volume is not necessarily a stand-alone entity, depending on what is on the volume. In particular, if the volume contains VSAM data sets of any type (including catalogs) then the volume contents are usable only if the complete VSAM metadata is available. Volume AAA may contain a VSAM data set that is cataloged on volume BBB. The VSAM information on volume AAA (including metadata in the VVDS) must be synchronized with the catalog information on volume BBB. In this simple example, migrating (copying) both volumes might suffice, provided the catalog on volume BBB is later properly connected to the master catalog on the target system.

In general, 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 includes all the relevant catalogs and VSAM data volumes.

A special warning is related to the migration utility described in this chapter. No enqueue or serialization functions are used by this utility. Transferring an active volume is probably not a good idea. If the volume is active, it might have logical errors when the transferred copy is used. For example, the 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 minidisk 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 DASD volumes containing minidisks.

## 15.2 Operational characteristics of the migration utility

The following characteristics of this utility are important:

- ▶ Complete volumes are transferred. This includes 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 simply copies and transfers all the tracks on the volume. It does not check whether the tracks are allocated (VTOC or VM equivalents) or in use (ENQ) or linked to a specific catalog (for VSAM, for example).
- ▶ A **read track** CCW is used to read the data on each track of a CKD volume. The amount of data on each track is variable. This means 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 z System, but this is not required. It could be a zPDT system. The receiving Linux side must be a Linux system with zPDT installed (but probably not active). The received copy of the source disk volume is stored in the awsckd (or awsfba if appropriate) format used by 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 reside 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 may be transferred in an unattended manner.
- ▶ TCP/IP port 3990 is used by default.<sup>2</sup> The port number may be changed when starting the server and client.
- ▶ Both 3380 and 3390 volumes, any size, can be migrated. This includes 3390 EAVs (“large volumes”).
- ▶ In practice, this utility is likely to run unattended because copying multiple volumes can take considerable time. We suggest you first try a single volume and monitor the operation while it is running.
- ▶ Only one transfer may be active for the server. It is possible to run multiple servers in parallel (with different IP port numbers for each) but the usefulness of this is questionable.
- ▶ The z/OS version is only for the migration of CKD DASD volumes. The z/VM version can migrate CKD and FBA DASD volumes.

## 15.3 Installation of the migration utility for z/OS

Several steps are required to install the migration utility:

1. Upload the server module (named ZOSSERV.XMIT) and install it in an authorized z/OS library. (The server module is provided in the `/usr/z1090/bin` directory that contains all the zPDT executables. It is an unloaded PDS member that has been processed by the TSO XMIT command.)

---

<sup>2</sup> 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.

2. Provide the required RACF definitions.
3. Determine whether TCP/IP port 3990 (on the z/OS side and the Linux side) has been 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

File `/usr/z1090/bin/ZOSSERV.XMIT` must first be uploaded (binary) to a z/OS data set with DCB characteristics `RECFM=FB, LRECL=80, 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 DSNs 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 ISPF to create these two holding data sets.)

After the receiving data set is ready, use binary mode and either `IND$FILE`<sup>3</sup> 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 TSO commands (using the appropriate data set names, of course):

```
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, once restored, is `ZPDTMSRV`. You must select an authorized library (preferably on the `LNKLST`) to contain the server module. You must have 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. Userid `IBMUSER` typically has update authority for all system libraries and we use this userid in our example. Again, your system may be different.

Copy member `ZPDTMSRV` from your holding PDS to your authorized library. The easiest way to do this 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)
```

<sup>3</sup> `IND$FILE` is a z/OS program that works with the "file transfer" function present with many 3270 emulators.

. ZPDTMSRV  
and Enter)

*(overtyp e the period with the letter S*

This completes the server installation. The holding data sets can be deleted.

### 15.3.2 RACF requirements

**Important:** Consult with the RACF administrator for your server z/OS system before making any RACF changes. This is especially important 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 by logging on to TSO with a userid having RACF SPECIAL authority and issuing these TSO commands from a READY prompt or ISPF option 6:

```
SETROPTS LIST                      (check the list of active classes)
RLIST DASDVOL *                    (see if any profiles are defined)
```

If these checks are negative, you can assume that the DASDVOL class is not being used.

#### DASDVOL not already 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 elect to potentially 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)
```

Be aware that the DASDVOL class is used only by a selected group of utility programs, such as dump/restore. Allowing UACC(ALTER) does not open all your data sets to access by all users. Whatever RACF data set protection you have in place (via ADDSD and PERMIT commands) is still effective.

This is all the RACF setup requires if DASDVOL was not initially active and if you want to allow the migration server to access any volume. If you want to limit the volumes 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 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 ADCD can be accessed by the migration utility.

The use 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 already in use

If you find that the DASDVOL class is active on your server system, we *strongly* suggest that you discuss the situation with the system programmers managing that system. Your requirements are simple: you (meaning the userid who will run the migration server program) need DASDVOL with READ access to whatever volsers you intend to migrate.

## TCP/IP port

By default, the migration programs use TCP/IP port 3990, but this can be changed. 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 could dynamically acquire the port. There is no easy way to prevent this. We suggest specifying a different port number only if there are error messages when you try to start the migration server or client. Someone must also verify that whatever firewalls are active will allow port 3990 communication.

# 15.4 Operation of the server under z/OS

The server should not be started until you are ready to use it. It automatically terminates after ten minutes of inactivity. The following JCL could 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 be used. We suggest port 3990 unless you have a specific reason to use a different port.
- ▶ The SYSTCPD statement must point to the DATA 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 is to examine the procedure used to start TCP/IP on your system.
- ▶ If your authorized library is not in LNKLIST, you will need a JOBLIB or STEPLIB statement that references only authorized libraries.
- ▶ The user submitting this job must have a userid that has at least READ access to the appropriate volsers in DASDVOL.

# 15.5 Installation of the server under z/VM

The z/VM system must be Release 5.4 or later.

The ZVMSERV.XMIT file is located with other zPDT binary material in /usr/z1090/bin on the zPDT system. This must be uploaded in BINARY, FIXED LRECL 80 format to a CMS user's A disk. (This file essentially contains a card deck.)

After this is complete, use 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. Beside the file in your RDRLIST, issue **RECEIVE** command.

This results in the reconstructed executable file on your CMS A disk, and completes the server installation.

## 15.6 Operation of 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 and this defaults to port 3990. When started, the server should indicate that it is waiting for a client connection. The server will time out after 10 minutes without a client connection.

If a z/VM mini-disk is migrated, it will appear as a small 3390 volume on the receiving system. That is, it will no longer be a mini-disk on a larger z/VM volume.

## 15.7 The client commands

There are two client commands:

- ▶ **hckd2ckd**: Used with both z/OS and z/VM to migrate a CKD DASD volume.
- ▶ **hfba2fba**: Used only with z/VM to migrate a FBA DASD volume.

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

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

Where:

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

:port is a TCP/IP port number to be used by both the client and server program. It defaults to 3990. **Note:** recent experience indicates the port number does not default correctly; we suggest you always include the port number in your command.

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

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

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

Either the -u or -v parameter must be supplied for DASD, 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 may be started on the Linux system. Remember that zPDT need not be operational for this. (We generally suggest that it not be operational, because the migration utility can place a heavy load on the LAN interface.) Examples of commands that could 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 have been started using the same port number, of course.) The second operand is the Linux file name 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 determine which volume is to be processed.

## 15.8 Additional notes

After a volume is migrated to Linux, you can add it to your devmap and access it from zPDT. You should check the permission bits for the file. (The zPDT system must have read/write access to it.) Our examples are in terms of z/OS volumes, but the volume could be for z/VM, z/VSE, or Linux for z Systems.

### Linux volumes

We discovered an interesting situation when migrating Linux for z System volumes. Our particular experience was with SLES-10 SP1 (for z System), but it might apply to other distributions.

With many Linux distributions, several *fstab options* can be selected for controlling disk volume mounts. These include mounting by device name, volume label, UUID, device ID, or device path. The default is often to mount by device ID. This produces a boot parameter list (and fstab) something like this:

```
parameters='root=/dev/disk/by-id/ccw-IBM.750000000M1881.2c23.1c-part1'
```

This disk identification is unique to the original disk drive and is useless when the volume is copied or migrated to another disk. In this situation, the migrated Linux volumes could not be booted. This identification is best changed when installing Linux by selecting the use of 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 additional 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 used to start the desired TCP/IP stack.

## Typical client usage

A typical use 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 consumed roughly 150 processor seconds on the source z/OS system (which was a different zPDT system on a moderate-performance PC).

When the migration is complete, the cylinder number displayed is one less than the actual number of cylinders transferred. Also, there might be a delay (perhaps up to 30 seconds) between the last message and the time the command ends. Both these conditions are normal.

## Migrating a list of volumes

Using **gedit** or another editor, you can create a Linux file named, **mig**, for example. The **mig** file contains 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                                (execute the commands in file named mig)
```

This transfers all the volumes that are listed, without any further manual intervention.



## Channel-to-channel

The `awsctc` device manager provides channel-to-channel (CTC) functions using TCP/IP communication paths. As shown in Figure 16-1, the connection can be within the same zPDT instance, between zPDT instances on the same machine, or between zPDT instances on different machines. Among other functions, CTC can be used by z/OS for GRS “rings”, NJE connections, XCF, TCP/IP connections, and so forth.

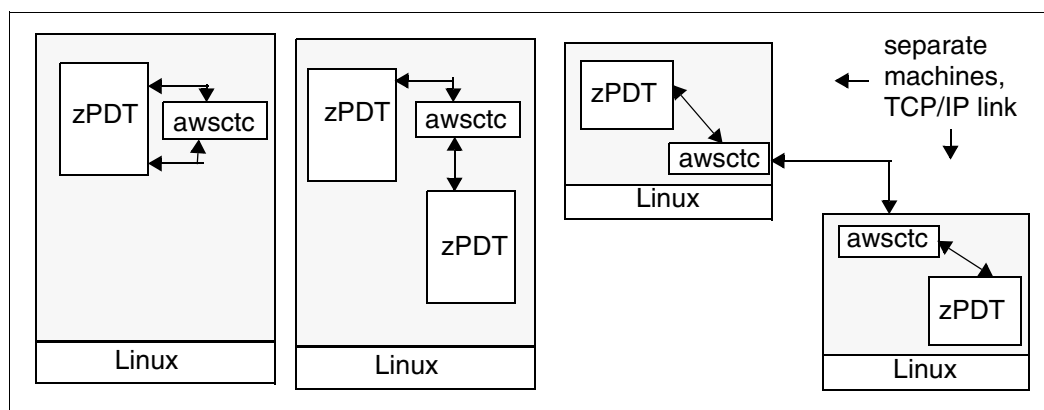


Figure 16-1 CTC links for zPDT

The `awsctc` device manager emulates IBM 3088-4 (or 3088-8) control units and devices.<sup>1</sup> The 3088 is an older product that provided the “middle” function for connecting two channels to each other. Each channel saw the 3088 as a control unit that provided a number of devices (each of which was a connection path). Modern systems have this control unit function integrated with the channels, and physical 3088 boxes are no longer used. However, the logical function has not changed.

The 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
```

<sup>1</sup> These are parallel channel control units. ESCON and FICON CTC operation are not emulated.

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.<sup>2</sup> 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, you must know how the devmap is defined on the other system. The same device number is often used at both ends, but this is not required.

An example of a simple connection between two zPDT instances in two different machines is shown in Figure 16-2.

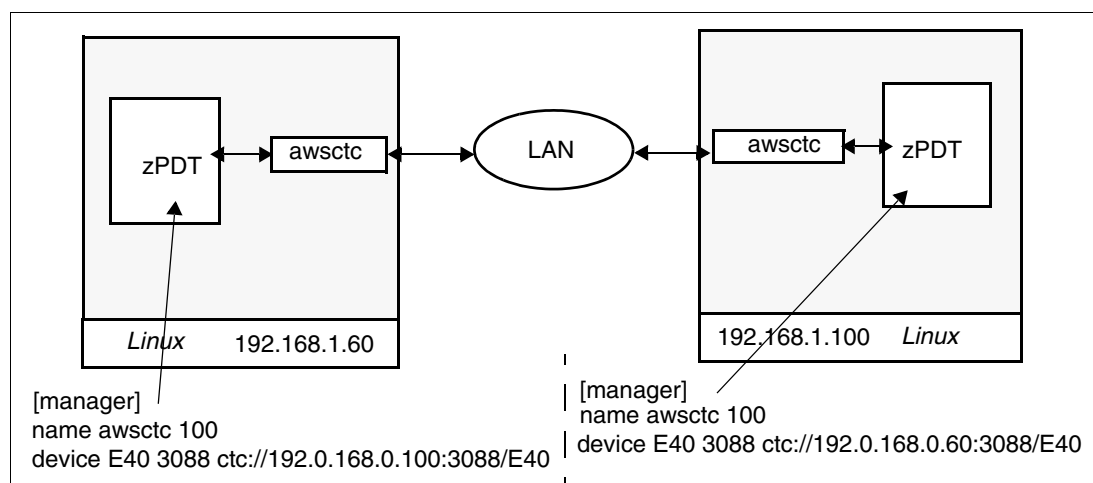


Figure 16-2 Simple two-system CTC definition

In this example, two separate zPDT instances are shown. Because they are separate instances, the CUNUMBR (100 in the example) can be the same in both definitions (but there is no requirement to do this, of course). Both instances elected to use the same device number (E40) for the CTC device, although there is no requirement to do this. 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 the "ctc://" is optional, if a protocol is specified it must be "ctc". The "well known" port numbers below 1024 cannot be used; the port number must be in the range of 1024 and 65535. The IP address string cannot be specified with an **awsmount** command.

Each CTC address must be defined. A real 3088 had multiple device numbers 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 do this. Each specific device number must be defined.

## Status display

The **awsstat** command may be used to display the status of the CTC device. When the device's peer is not yet 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 generally does not make any

<sup>2</sup> We use ports 3088 and 3089 for these examples because it is easy to remember. There is nothing special about this port number.

difference to the user. After data begins to flow, the Accepted/Connected is shortened to A or C and the number of send/receive bytes is displayed.

## 16.1 z/OS use example

Configurations using CTC can be complex. We have taken a basic NJE example, as shown in Figure 16-3.

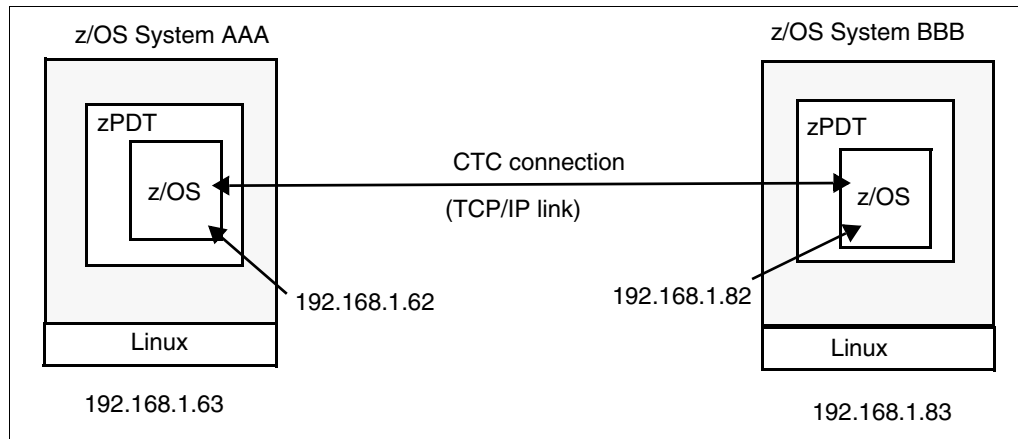


Figure 16-3 Trivial NJE setup

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 is true for current z/OS AD-CD systems.) Be careful to use the IP address of the remote Linux system (and not the remote z/OS system) in the devmaps.

The devmap for system AAA might contain the following stanza:

```
[manager]
name awscctc 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, of course)
```

The device status usually starts as connecting or accepting, and later switches to connected or accepted. This may 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 IPL ing z/OS. The link can be checked with the **awsstat E40** command (using the correct device number for your CTC, of course). This normally takes only a few seconds, but can take longer in rare cases. We then needed to issue commands on each JES2 to make the connection operational:

```
$$ N,LINE1 (sometimes needed)
$$ LINE1
```

A job submitted on system AAA could be directed to system BBB for execution 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 (very useful status display)
$$ LINE(1)
$$ N,LINE1 (discover dynamic connections)
$D NODE(*)
$D LINE
$D DESTID(*)
$D PATH(*)
$P LINE(1) (stop line)
$E LINE(1) (reset line)
$D NJEDEF (display NJE definitions)
$N,D=BBB, '$D NJEDEF' (send JES2 command to other node)
```

A JES2 cold start may be necessary to enable NJE changes to JES2PARM. You need to stop the line, **\$P LINE(1)**, when stopping JES2. If the **\$P** is not effective, try **\$E LINE(1)** followed by **\$P LINE(1)**.

## 16.2 Multiple instances and z/VM

The following material outlines several uses of CTC connections with two instances of z/VM.<sup>3</sup> This material is intended as a general overview and does not provide complete step-by-step instructions for implementation and usage. It is very unlikely that all the links shown in this example will be present in any practical system.

### 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
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 TCPIP
device 700 3088 3088 ctc://localhost:3701/701      #for TCPIP
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
```

---

<sup>3</sup> Thanks to Bruce Hayden, of the Washington Systems Center, for this material.

### ***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 TCPIP
device 700 3088 3088 ctc://localhost:3701/701  #for TCPIP
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**

Notice that each instance has its own copy of the z/VM disks; they are in separate directories: /z1 and /z2.

#### ***ISLINK***

An ISLINK creates an ISFC (Inter-System Facility for Communications) connection. It is the easiest link because it involves only CP commands and does not require a virtual machine or userid. The VM system identifiers of the two connected systems must be different. Using the devmaps 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 should result in the message HCPALN2702I Link 0720 came up. The command **Q ISLINK** can be used to display the status of the connection.

#### ***TSAF***

TSAF is an older function and probably would not be used if ISFC is available. To try it, use commands such as the following (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**.

#### ***RSCS***

RSCS can be more complicated. The RSCS product is not enabled by default. You can check this with the command **Q PRODUCT STATE ENABLED**. If RSCS is not in the list, it can be enabled (using MAINT) by 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 **MSG RSCS START IBMSYS1** on ibmsys2 and **MSG RSCS START IBMSYS2** on ibmsys1. The status of the RSCS connection can be displayed with **MSG 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 this you must assign host names and domain names; in an isolated environment these can be arbitrary names. For an isolated environment with only two nodes the gateway address does not matter; you might use 10.1.1.1.<sup>4</sup> 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. The status of TCP/IP can be checked by:

```
VMLINK TCPMAINT 592
NETSTAT DEVL
```

You can force TCP/IP to restart with the following commands:

```
FORCE TCPIP
XAUTOLOG TCPIP
```

<sup>4</sup> Remember that the default address of the tunnel connection to the base Linux is 10.1.1.1. This default is used in the sample devmap.





# Cryptographic usage

A zPDT system can emulate multiple cryptographic adapters as CEX7S devices.<sup>1</sup> Each CEX7S emulated adapter runs as separate Linux processes. If sufficient base processor cores are available to permit these threads to be dispatched in parallel by Linux, the emulated adapters can run asynchronously with the zPDT CPs.

Do not confuse a cryptographic adapter with the cryptographic instructions that are always available with a zPDT system. These are the KM, KMC, KIMD, KLMD, KMAC, and associated instructions (known as the CPACF instructions) that provide a number of fundamental cryptographic operations.

The cryptographic instructions may be coded directly in a program or used through ICSF<sup>2</sup> programming interfaces. For practical purposes, the cryptographic coprocessor facilities are available only through ICSF programming interfaces.

**Important:** The Cryptographic adapter 7S emulation included with zPDT GA10 uses a different format for master keys than that used with earlier adapters. zPDT GA10 does not automatically convert the earlier format to the current format. You must initialize the GA10 cryptographic adapter with the same master keys previously used if you need to access previously encrypt data.

## 17.1 Background information

A cryptographic coprocessor is represented by an adjunct processor (AP) process in zPDT. A System z CP sends and receives work entries through coprocessor queues that are known as *domains*.

The *accelerator mode and customized (UDX)* functions of a cryptographic coprocessor are not provided for zPDT. Trusted Key Entry (TKE) systems, which are personal computers with unique software and an appropriate cryptographic adapter, are not used with zPDT. PKCS#11 is not available with zPDT emulation.

<sup>1</sup> This assumes using zPDT GA10 or later. Earlier releases of zPDT had earlier levels of CEX functionality.

<sup>2</sup> This is the Integrated Cryptographic Service Facility.

Each emulated cryptographic coprocessor has 16 domains;<sup>3</sup> each z/OS instance uses a different domain (this is specified in the ICSF parameters). If multiple coprocessors are defined, then z/OS uses the same domain number in each coprocessor. If multiple coprocessors are defined, z/OS can dispatch requests to all of them (using the same domain number in each processor).

With zPDT, the cryptographic coprocessor keys (and other state information) are stored in the ~/z1090/srdis directory. There is a separate subdirectory for each defined coprocessor. These directories are created automatically by zPDT.

Any tampering with this information can produce unpredictable results. However, a complete coprocessor subdirectory may be deleted as a way of reinitializing (zeroing) that coprocessor. (The **ap\_zeroize** command, described later, is another way to reinitialize a coprocessor.)

It is important to understand that zPDT cryptographic functions are intended for development purposes and not for security. While the format of the key data in ~/z1090/srdis is not documented, it is certainly not secure in any cryptographic sense.

## 17.2 Devmap specification

The zPDT specification for emulated cryptographic adapters is part of the 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 .....
```

The zPDT architecture allows up to 16 or 64 cryptographic coprocessors, depending on the overall configuration.<sup>4</sup> z/OS allows a maximum of 16.

## 17.3 Initial ICSF startup

For practical purposes, the ICSF software functions are needed to initialize cryptographic adapters. The following is a brief outline of the steps we took to customize and use the ICSF panels on a z/OS AD-CD 2.4 system. The use of the AD-CD system PARMLIB and PROCLIB is not required, of course, so adjust these names for your needs.

Recent AD-CD z/OS releases automatically start CSF and have the basic customization to use both the CPACF instructions (which are always enabled) and CEX7S functions (if defined in your devmap). The most recent z/OS AD-CD (at the time of writing) uses ICSF level HCR77D1.

<sup>3</sup> CEX7S functionality on a larger System z has more domains; zPDT is limited to 16 domains.

<sup>4</sup> A zPDT group controller instance may have up to 64 coprocessors defined, otherwise the limit is 16.

We found that we needed to specify a domain number in the PARMLIB member associated with ICSF:

In the PARMLIB member CSFPRM00 (this example is from the June 2018 AD-CD):

```
CKDSN(CSF.CSFCKDS)
PKDSN(CSF.CSFPKDS)
COMPAT(NO)
DOMAIN(0)                <-- Specify a domain number
SSM(NO)
CHECKAUTH(NO)
CTRACE(CTICSF00)
USERPARM(USERPARM)
REASONCODES(ICSF)
```

► In the PARMLIB member IKJTSO00:

```
AUTHPGM NAMES(          /* AUTHORIZED PROGRAM NAMES      */ +
...
CSFDAUTH          /*THIS WAS ALREADY IN OUR AD-CD SYSTEM */ +
CSFDPKDS         /*WE ADDED THIS ONE                */ +

AUTHTSF NAMES     /*PROGRAMS.....                      */ +
....
CSFDAUTH          /*THIS WAS ALREADY IN OUR AD-CD SYSTEM */ +
CSFDPKDS         /*WE ADDED THIS ONE                */ +
```

Be certain to copy the plus signs (+) at the end of each line! These additions should prevent 047 ABENDs when customizing the cryptographic coprocessor.

To begin customization of a cryptographic coprocessor, go to ISPF option 6 and enter the command **@ICSF**. This should produce the first ICSF panel, similar to that shown in Figure 17-1.

```
HCR77B0 ----- Integrated Cryptographic Service Facility-----
OPTION ==>
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
5694-A01 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.
Press END   to exit to the previous menu.
```

Figure 17-1 First ICSF panel

On this panel, select option 1. This should produce a display similar to Figure 17-2. This panel verifies that the coprocessor is active. The “5C00” nomenclature varies with releases.

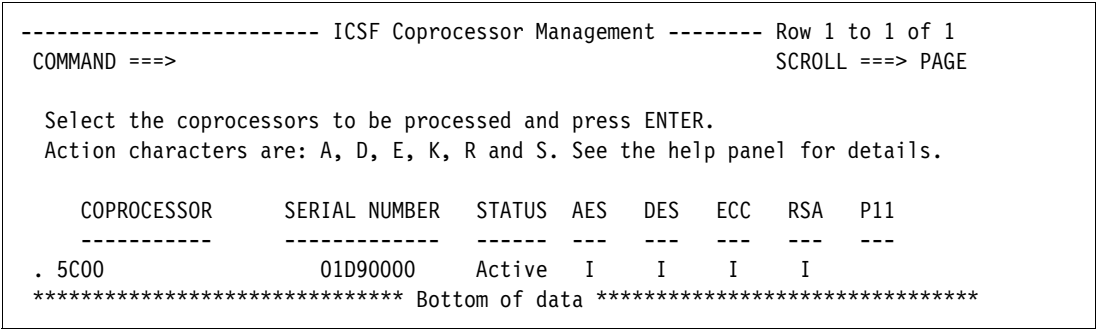


Figure 17-2 Verify that the cryptographic coprocessor is online

Use F3 to return to the first ICSF panel and select option 6. This option allows you to enter a *pass phrase* that is then automatically used to initialize the basic coprocessor 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 coprocessor management, we suggest that you use the simple pass phrase initialization process we describe here.)

Complete the pass phrase panel as shown in Figure 17-3 (using your own pass phrase, of course). You need to enter your pass phrase, two data set names, specify no KDSR format, and a character to select the **Initialize system** option. The two dataset names shown in the example (CSF.SCSFCKDS and CSF.SCSFPKDS) are preallocated but empty in the current AD-CD z/OS system.

The sample jobs to allocate these datasets 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, provided 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 Pass phrase panel*

Start ICSF again and select option **1**. Overtyping the initial period in front of the 5C00 coprocessor name with the letter **S** and pressing Enter produces a display similar to that shown in Figure 17-4 on page 312; your details will differ but you should see that the current master key for CKDS and PKDS is VALID.

```

----- ICSF - Coprocessor Hardware Status -----
COMMAND ==>                                SCROLL ==>
                                           CRYPTO DOMAIN: 0

REGISTER STATUS                                COPROCESSOR  5C00

                                           More:      +

Crypto Serial Number      : 01D9T740
Status                    : ACTIVE
AES Master Key
  New Master Key register : EMPTY
  Verification pattern    :
  Old Master Key register : EMPTY
  Verification pattern    :
  Current Master Key register : VALID
  Verification pattern    : BCB28DF1FA0FC8EF
DES Master Key
  New Master Key register : EMPTY
  Verification pattern    :
  Hash pattern           :
                          :
  Old Master Key register : EMPTY
  Verification pattern    :
  Hash pattern           :
                          :
  Current Master Key register : VALID
  Verification pattern    : 2D7BDF2F5A7ADF9C
  Hash pattern           : 071B3CB4761DCDE4
                          : E1D3675B90E977C7
ECC Master Key
  New Master Key register : FULL
  Verification pattern    : xxxxxxxxxxxxxxxxx
  Old Master Key register : EMPTY
  Verification pattern    :
  Current Master Key register : VALID
  Verification pattern    : DB17D4FE5087CFE8
RSA Master Key
  New Master Key register : EMPTY
  Verification pattern    :
                          :

Press ENTER to refresh the hardware status display.
Press END   to exit to the previous menu.

```

Figure 17-4 Coprocessor status after setting pass phrase

This completes the basic cryptographic coprocessor setup. We suggest you do not experiment with option 2 (MASTER KEY MGMT) functions unless you are certain you know what you are doing.

## 17.4 Operational notes

CSF must be started (as a started task) in order to use it. This is automatically done in current AD-CD z/OS releases. You can easily verify 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 zPDT cryptographic coprocessor emulation functions are intended for use by developers who require these functions. It should be clearly understood that, while using zPDT, these emulated functions are not intended to produce a secure system or to function as a secure peer when dealing with private data.

If you used a pass phrase for initialization, record the exact characters used (including upper or lower case, spaces, and punctuation). You need this to recreate the same master keys if you reinitialize the cryptographic functions or want to create duplicate keys on another System z.

### 17.4.1 Multiple zPDT instances

zPDT may have multiple instances in operation. These instances may have shared facilities, such as shared DASD. If shared facilities are used, then a zPDT controller instance<sup>5</sup> must be present as described in Chapter 10, “Multiple instances and guests” on page 197. Coprocessors defined in the controller instance may be shared by all zPDT instances. A maximum of 16 coprocessors may be present in a “normal” zPDT instance. A maximum of 64 coprocessors may be defined for a controller instance.

An additional devmap statement:

```
domain <member name> a y    #{a is a coprocessor number, y is a domain number}
```

is used in the devmap of a zPDT instance that is using coprocessors defined in the controller instance. The member name is required if the domain statement appears in the controller instance; it 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 separated by commas, or a range of numbers separated by a dash. The angle brackets around the member name are not part of the syntax; they indicate an optional parameter here. Remember that the domain numbers must be specified in the ICSF startup parameters and are different for each z/OS instance/

Three shared cryptographic coprocessors, used by three 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
```

<sup>5</sup> Very briefly, a controller instance is a zPDT instance (with a separate devmap and started with an **awsstart** command) that does not contain any System z processors.

```

#----- 1090 instance 1 -----
[system]
processors 1
...
[adjunct-processors]
domain 0 1
domain 1 1
domain 2 1

#----- 1090 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.

#----- 1090 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.

```

Notice that each 1090 instance has a different domain number specified in the domain statements. In this example the domain numbers are the same as the instance numbers, but this is just a coincidence. zPDT supports a maximum of 16 domains for each emulated coprocessor.

## 17.4.2 Coprocessor control commands

A number of commands are included for specialized management of the cryptographic coprocessors. These commands are issued from a Linux terminal window. zPDT must be operational for these commands to be used. *They are not needed for normal system use* and we suggest you do not experiment with them unless you have a fairly good understanding of what you are doing. In the following commands the *n* variable is the cryptographic coprocessor 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 coprocessor. 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 the coprocessors available to the System z. With an operand, it lists which domains are used by the indicated coprocessor.

```
$ ap_query
```

- ```
$ ap_query -a n
```

  - This command creates a new (emulated) cryptographic coprocessor.
- ```
$ ap_create -a n
```

  - This command removes the indicated coprocessor process if it is not connected to a CP process.
- ```
$ ap_destroy -a n
```

  - These commands vary online or vary offline connections between coprocessors 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 for the indicated coprocessor.
- ```
$ ap_vpd -a n
```

When a zPDT instance is started (while processing the devmap) an **ap\_create** is automatically issued for that instance. If this is a stand-alone zPDT instance, **ap\_von** commands are issued for all domains. (It is not issued for a controller instance.) If this is a zPDT instance using shared coprocessor resources, **ap\_von** commands are issued for the coprocessors and domains specified in the devmap.

The “real” cryptographic coprocessors on large System z 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 coprocessor master keys, stored in the Linux `srd` subdirectory, must be consistent with the data in the CSF.SCSFCKDS and CSF.SCSFPKDS data sets in z/OS. If you install a new z/OS release and create new 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.

For long-term cryptographic usage, you should place the CSF.SCSFCKDS and CSF.SCSFPKDS data sets on local volumes that will be used with all the releases of z/OS that you might want to invoke.

If you plan to work with encrypted data (as opposed to simply developing programs that use encryption functions) you need to carefully plan backups for the coprocessor data (in the `srd` subdirectory) and the z/OS data sets used by CSF. The zPDT functions have no special way to recover lost encryption keys.

### 17.4.4 Programming with CSF

Following is a trivial program that uses the cryptographic coprocessor (via an CSF programming interface) to obtain random numbers:

```
//OGDENYZ JOB 1,OGDEN,MSGCLASS=X
//A EXEC ASMACLG,PARM.C='NOXREF',PARM.L='NOLIST,NOMAP'
//C.SYSIN DD *
          PRINT NOGEN
ICSFAA   CSECT
ICSFAA   AMODE 31
```

```

ICSFAA  RMODE  24
        STM    14,12,12(13)      SAVE CALLER'S REGISTERS
        LR     12,15             USE ENTRY-POINT BASE REGISTER
        USING  ICSFAA,12
        LR     2,13             GET A(CALLER'S SAVEAREA)
        LA     13,SAVEAREA       GET A(MY SAVEAREA)
        USING  SAVEAREA,13      MORE 'USING' SPACE
        ST     2,SAVEAREA+4     CHAIN OLD TO NEW
        ST     13,8(2)          CHAIN NEW TO OLD

*
* OPEN FILES AND CHECK RESULTS
*
A1      OPEN   (PRINTD,(OUTPUT))
        TM     PRINTD+48,X'10'   CHECK SYSPRINT OPEN STATUS
        BZ     ERROR1

*
* 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
CLOSEALL CLOSE (PRINTD)
RETURN  L      13,4(13)         GET A(CALLER'S SAVE AREA)
        LM     14,12,12(13)     RESTORE CALLER'S REGISTERS
        SR     15,15           SET RETURN CODE
        BR     14             EXIT

*
* SIMPLE ERROR HANDLING.
*
ERROR1  WTO    'UNABLE TO OPEN SYSPRINT DD STATEMENT'
        B      RETURN

*
ERROR2  WTO    'NON-ZERO RETURN CODE'
        B      RETURN

*
PRINTD  DCB    DSORG=PS,MACRF=(PM),DDNAME=SYSPRINT,LRECL=80,      X
        RECFM=FB,BLKSIZE=8000

* 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

*
DROP 12

```

```

SAVEAREA DC 18F'0'
SW1      DC D'0'      WORK AREAS FOR UTILITY ROUTINES
SW2      DC D'0'      WORK AREA
SPILL    DC D'0'      SPILL FROM SW2 UNPK INSTRUCTION
SWOUT    DC CL80' '    OUTPUT AREA
SBLANKS  DC CL80' '    SOURCE OF BLANKS
SZEROS   DC 2F'0'      SOURCE OF ZEROS
ASCNDECS DC 8F'0'      LOCAL REGISTER SAVE AREA
          SPACE
*-----
* FORMAT 32 BYTES OF STORAGE INTO HEX.
* INPUT: R1 CONTAINS ADDRESS OF DATA. OUTPUT: 72 BYTES IN SWOUT
*-----
AHEXLINE STM 1,6,ASCNDECS    SAVE CALLER'S REGS
          LA 2,8            8 WORDS OUTPUT
          LA 3,SWOUT        A(OUTPUT)
          MVC SWOUT(80),SBLANKS
AHEXLINF BAL 5,AHEXLINZ      CONVERT 4 BYTES
          LA 3,9(3)         OUTPUT POINTER
          LA 1,4(1)         INPUT POINTER
          BCT 2,AHEXLINF     LOOP
          LM 1,6,ASCNDECS
          BR 10             RETURN TO CALLER
*
AHEXLINZ MVC SW1(4),0(1)
          MVI SW1+4,X'00'
          UNPK SW2(9),SW1(5)
          TR SW2(8),AHEXTR-240
          MVC 0(8,3),SW2
          BR 5
AHEXTR   DC C'0123456789ABCDEF'
          SPACE
          LTORG
          END
/*
//L.SYSLIB DD DISP=SHR,DSN=SYS1.MACLIB
//          DD DISP=SHR,DSN=CSF.SCSFMODE
//G.SYSPRINT DD SYSOUT=*

```

## 17.5 Very basic dataset encryption example

Many zPDT or zD&T users also use the AD-CD z/OS package. Based on zPDT release GA8 (December 2017) and the June 2018 z/OS 2.3 AD-CD, the following instructions provide two options for simplistic implementations of dataset encryption.<sup>6</sup> One option uses RACF and a specific HLQ to select datasets for encryption, and the other option uses JCL parameters to control encryption. The setup is essentially the same for both options.

*These examples ignore all the planning and controls that would be used in a production environment and concentrates only on a basic setup for dataset encryption.* There are many ways to accomplish the various steps needed; the following instructions outline one of these ways.

<sup>6</sup> Later z/OS AD-CD release might already include some of the RACF work described in this section.

This example assumes a basic systems programming familiarity with z/OS and the z/OS AD-CD systems. The following names are used in the example:

- PROT is an arbitrary choice, used both as an HLQ and as a Storage Class name; there is no need to use the same name for both, this was also arbitrary.
- PROTECT is an arbitrary choice, used as a Storage Group name
- SYS1.S0W1.SCDS and SYS1.SMS.CNTL exist in the current z/OS AD-CD
- DB2STORG and DB2STORC are member names used in the current z/OS AD-CD.
- ADCDA and ADCDB are userids (in the current AD-CD) with no privileges.
- The address (AB0) used for a new volume is arbitrary within AD-CD IODF constraints.
- The new volume volser (SMS001) is arbitrary.
- MYPROT.KEY1 is an arbitrary label for an encryption key.
- PROT.TRY1 is an arbitrary dataset name with HLQ PROT.
- OGDEN.TRY2 is an arbitrary dataset name without a special HLQ.
- The Linux directory /z/zos23 represents the working directory for running zPDT, but users might have their zPDT in any directory.

One goal is to use RACF controls to automatically encrypt any dataset 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). Userid ADCDA is allowed access to the key and the datasets. Userid ADCDB is allowed access to the datasets but not the key.

### **Volume setup**

You need a DFSMS-managed volume for extended datasets because, at the time of writing, this is the only category that can be encrypted with the "dataset encryption" functions. Create this volume, before starting zPDT, with the zPDT command:

```
$ alckd /z/zos23/SMS001 -d3390-1    (choice of a 3390-1 is arbitrary)
```

Add this volume to your 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/zos23/SMS001
...
```

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
/*
```

Vary the new volume online.

## **SMS setup**

We need to make the new SMS-managed volume usable. This involves defining a Storage Group (containing the new volume) and a Storage Class for the target datasets. These are the minimum SMS controls required. The following is an abbreviated description of the steps needed while working with ISMF.

- a. Start ISMF. If you cannot see option 6 then do the following: option 0 (ISMP Profile), option 0 (User Mode), option 2 (Storage Administrator). Exit from ISMF and start it again.
- b. Create a Storage Group named PROTECT.  
Select option 6 (Storage Group) and enter 'SYS1.S0W1.SCDS' as the CDS Name. Enter PROTECT as the Storage Group Name, POOL as the type, and Select option 3 (Define). Change any of the characteristics that appeal to you. We changed only Guaranteed Backup Frequency to NOLIMIT because ISMF seemed to want an entry for this parameter.  
Next, select option 5 (Volume) then option 2 (Define). Enter your selected volser (SMS001 in this example) in a prefix line.
- c. Create a Storage Class named PROT.  
Select option 5 (Storage Class) and enter 'SYS1.S0W1.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.
- d. Update the Storage Class and Storage Group ACS routines.  
Select option 7 (Automatic Class Selection). Enter 'SYS1.S0W1.SCDS' as the CDS Name and Select option 1 (Edit). This produces the ISPF edit panel. Enter Name 'SYS1.SMS.CNTL(DB2STORG)' as the dataset 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
```

Save (PF3) and then edit 'SYS1.SMS.CNTL(DB2STORC)' adding the following lines:

```
WHEN (&HLQ = 'PROT')
DO
    SET &STORCLAS = 'PROT'
    EXIT
END
```

Go to the end of the DB2STORC member and look for an OTHERWISE statement.

```
OTHERWISE
DO
    SET &STORCLAS = &STORCLAS      (it was &STORCLAS = '')
    EXIT
END
```

Exit the edit panel and select 2 (Translate) in the ACS Application panel. Set 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 Enter. If the results are good, repeat the process for member DB2STORC. (The statements after OTHERWISE

in DB2STORC are odd; they are needed to accept a storage class entered in a JCL DD statement.)

- e. Update the Control Data Set. Select ISMF option 8 (Control Data Set). CDS name is 'SYS1.S0W1.SCDS'. Select option 5 (Activate). Place a '/' character as indicated and Enter.
- f. Exit ISMF

### **RACF work**

Do the following RACF work while working as IBMUSER:

- a. 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)*

- b. Check access to RACF class CSFSERV, using ISPF option 6:

**RLIST CSFSERV \***

We found the access was UACC(READ), which is perhaps not a good idea.

**RALTER CSFSERV \* UACC(NONE)**  
**SETROPTS RACLIST(CSFSERV) REFRESH**

- c. Userids ADCDA and ADCDB might need their passwords reset.

**ALU adcda PASS(xxxxxx) NOEXPIRED** *(use appropriate passwords)*

**ALU adcdb PASS(yyyyyy) NOEXPIRED**

- d. Dataset 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 AD-CD we found this was already set for IBMUSER. You can verify this with:

**RLIST FACILITY STGADMIN.\***

You need to PERMIT users (other than IBMUSER) wanting dataset encryption to this profile.

- e. Working as IBMUSER in ISPF option 6, 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 ICSF*  
**SETROPTS RACLIST(CSFSERV) REFRESH**

As the profile owner, IBMUSER has ALTER access to these profiles.

- f. While you are working with RACF, do the following to enable automatic encryption for the HLQ we selected (PROT) and to allow access to the key we 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(SYMPACFWRAP(YES) -**  
**SYMPACFRET(YES))**

**PE MYPROT.KEY1 CLASS(CSFKEYS) ID(ADCDA) ACCESS(READ)**

**SETROPTS RACLIST(CSFKEYS) REFRESH**

The PERMIT statements to allow ADCDA and ADCDB access to PROT.\* datasets are not really needed because there have been no RACF profiles to prohibit their access. However, this illustrates what would probably be done in a more general environment. Remember that 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 308 if you have not already done this.

### **Create new encryption key**

Create a random secure key using a batch job.

```
//KEY 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)
/*
```

Check the job results. You can verify that the key exists with a simple IDCAMS job.

```
//IDCAMS JOB 1,OGDEN,MSGCLASS=X
// EXEC PGM=IDCAMS
//IN1 DD DISP=SHR,DSN=CSF.SCSFCKDS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
      PRINT INFILE(IN1) DUMP
/*
```

You next need to refresh the in-storage CKDS. Go to ICSF again, select option 8 and then option 4 which is REFRESH. Enter 'CSF.SCSFCKDS' (with the single quotes) as the New CKDS parameter and press Enter. (The title is slightly misleading because CSF.SCSFCKDS is not new in this case.) Exit ICSF.

### **Test and verify encryption**

Test RACF-controlled encryption with a job similar to this:

```
//TRY1 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,DSN=EXTREQ
```

Submit the job. If it runs without errors, try to browse the new PROT.TRY1 dataset. You should be able to browse it because it is automatically decrypted for IBMUSER (because IBMUSER has access to both the dataset and the key).

Logon as ADCDA and browse it. This should work because ADCDA has access to both the dataset and the key. Logon as ADCDB and browse it. This should fail because ADCDB does not have access to the key.

Is it really encrypted? As IBMUSER, run the following:

```
//ADRD 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)
/*

```

This should show the dataset as it is actually stored (encrypted) on disk.

Instead of using RACF to control encryption (via a preselected set of HLQs), we can specify encryption using only JCL control. In this case we avoid the HLQ of PROT in our example.

```

//TRY2 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,DSN=EXTREQ,
//          STORCLAS='PROT',DSKEYLBL='MYPROT.KEY1'

```

You can again verify the encryption by dumping the dataset with ADDRDSU. You might also use TSO commands such as the following to verify that the catalog entry denotes encryption and the key label.

```

LISTC ENTRIES('OGDEN.TRY2') ALL
LISTC LEVEL(PROT) ALL

```

### Tip

While experimenting with your sandbox zPDT system you might want to try initializing master keys again. You may find this difficult because the CKDS/PKDS datasets already have contents and a master key already exists. We used the following steps to clear everything.

- Stop CSF with a **P CSF** command.
- Use a zPDT command (entered from Linux) to clear the master keys:  
`$ ap_zeroize -a 0 -d 0` *(assume crypto adapter 0 and domain 0)*
- You might need access to a specific RACF profile:  
**RDEF CSFSERV CSFRSWS UACC(NONE)** *(as the owner, you have ALTER access)*  
**SETROPTS RACLIST(CSFSERV) REFRESH**
- Using ISPF 3.4, delete VSAM clusters CSF.SCSFCKDS and CSF.SCSFPKDS. (Using a **DEL** line command instead of a **D** line command is recommended.)
- Allocate new datasets with a job similar to 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(A3SYS1) -
  RECORDS(200 100) RECORDSIZE(252 252) KEYS(72 0) -
  FREESPACE(10 10) SHAREOPTIONS(2,3))
DATA (NAME(CSF.SCSFCKDS.DATA) BUFFERSPACE(100000) ERASE WRITECHECK)
INDEX (NAME(CSF.SCSFCKDS.INDEX))
DEFINE CLUSTER(NAME(CSF.SCSFPKDS) VOLUME(A3SYS1) -
  RECORDS(200 100) RECORDSIZE(800 3800) KEYS(72 0) -
  FREESPACE(0 0) SHAREOPTIONS(2,3))
DATA (NAME(CSF.SCSFPKDS.DATA) BUFFERSPACE(100000) ERASE WRITECHECK)
INDEX (NAME(CSF.SCSFPKDS.INDEX))
/*

```
- Start CSF (**S CSF**) and continue with the master keys as outlined above.

## 17.5.1 z/VM usage

Cryptographic coprocessors 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                 (coprocessors to be dedicated for use)
  OPTION TODENABLE MAINTCCW
  MACHINE ESA 64
  CPU 0 BASE
  CPU 1
  IPL 190 PARM AUTOCR
```

etc





# Virtualization and containers

zPDT may be used in a virtual environment or in docker containers.

## 18.1 Docker

IBM has done basic testing of zPDT running in a *docker* environment, indicating that this can be a reasonable environment for zPDT operation. As with other types of virtualization, we caution against overcommitment of the base system or understating a reasonable zPDT configuration. These cautions are usually related to memory sizes and availability.

Simple docker usage typically involves applications represented by a single Linux process. zPDT operation has many Linux processes; furthermore, zPDT operation requires the availability of auxiliary Linux programs such as the **awsckd** and **aws3274** device managers, and the **awsstart** and **ipl** commands. Network configuration can be especially complex to implement.

zPDT does not provide a sample docker script or image. If you build your own image you should consider the following requirements if you might seek IBM assistance for zPDT (in the container) issues:

- ▶ The Linux window used to start zPDT should be available for debugging.
- ▶ The 3270 window for the z/OS master console should be available for debugging.
- ▶ The container should expose the zPDT logs directory to the base Linux.
- ▶ We suggest including the Linux gbx package in the image.
- ▶ Any zPDT core image files should be exposed to the base Linux.
- ▶ These console windows and exposed files or directories must be available for all running containers.

## 18.2 Virtual environments

Three virtual environments have been used during zPDT development:

- ▶ VMware
- ▶ KVM
- ▶ Xen

Other virtual environments might operate correctly, but have not been tried by zPDT developers. In particular, the VMware Player (operating under Microsoft Windows) has not been investigated and no support is available for this configuration.<sup>1</sup>

Many options are available in a virtual environment. In some cases it is reasonable to substantially *overcommit* a virtual server; that is, to run virtual guests that (in total) could use more memory or more processor cycles or more I/O bandwidth than are actually available. This is typically done when the system administrator knows that the actual workloads are such that the overcommitment has no undesirable effects.

zPDT (running z/OS) is typically a “heavy” workload. *We strongly advise that you do not run 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 virtual machine hypervisor and the guest Linux running zPDT and z/OS might all be paging due to several levels of overcommitted memory.

As a general rule, you (the zPDT owner, user, and/or administrator) must obtain the necessary skills to install, configure, and use your virtual server. We do not attempt to document or provide instructions for installing and managing the virtual server environment.

## Observed problems

We have encountered a number of cases of virtual zPDT operation in servers managed by staff who are not familiar with the workload characteristics of zPDT, z/OS, and so forth. In some cases the server is overcommitted, with the server staff commenting that “we always plan this way.” Insufficient resources for zPDT can produce a variety of problems that are difficult to diagnose.

## License servers

The tested virtual environments normally used remote zPDT license and UIM servers. This might not be necessary for smaller configurations where a separate USB port (for a token) could be assigned to each virtual guest zPDT.

A single zPDT token (connected to a USB port) cannot be shared by multiple virtual machines. In general, the first virtual machine started (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 zPDT instances (assuming the token can provide sufficient licenses).

## Disk configurations

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

## Performance

We found 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.

<sup>1</sup> This does not mean that it fails. We simply did not investigate it and cannot offer any support for using it with zPDT.

The key consideration is the nature of the workloads. The focus in this document is z/OS, but this 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, whereas another z/OS system with only a few users running large Db2 or Java jobs might be quite heavily loaded. You cannot draw any conclusions about performance unless you can realistically define your workloads.

Our notable points included these items:

- ▶ If the z/OS workloads tended to drive z/OS close to 100% CP usage, then overcommitment of server cores resulted in poor performance. Performance degradation was not linear. When the processors were overcommitted (with near 100% utilization by z/OS workloads) performance dropped dramatically.
- ▶ z/OS jobs with heavy I/O ran considerably slower than in a non-virtual environment, even with no overcommitment of cores. Some MIH<sup>2</sup> messages were seen, but z/OS recovered from these.
- ▶ Light z/OS loads, such as TSO use with occasional compilations, ran very well across multiple virtual machines.
- ▶ We did not overcommit memory. This is possible with some hipervisors, but we avoided it. Such overcommitting of memory could result in paging at hipervisor level and we assumed this would degrade overall performance.
- ▶ Except for very light workloads, most z/OS jobs (in virtual machines) ran a little slower than in non-virtual environments. This was expected and accepted. What was unexpected (although reasonable) was the apparent processor time (TCB + SRB times) for z/OS jobs was longer than in a non-virtual environment. That is, the apparent System z instructions ran slower, and thus needed more System z processor time. This might be significant in a situation where processor time (as reported by SMF) is important for some reason.
- ▶ Each virtual machine (for zPDT) had a base Linux. As usual, the Linux disk cache is a critical performance factor. Using monitoring facilities we noted that the complete virtual machine memory was heavily used. We suggest that giving more memory to each virtual machine than you might provide in a non-virtual environment (without overcommitting memory) may improve performance. In effect, this allows more buffering of I/O to the real disks on the server.

---

<sup>2</sup> MIH is missing-interrupt handler. This is a z/OS function that can be triggered by very slow (or backlogged) I/O operations.





# Problem handling

There are several aspects to handling zPDT problems, including these:

- ▶ Problems starting a zPDT operation
- ▶ Problems during a zPDT operation
- ▶ Problems with emulated device files
- ▶ Problems with the underlying Linux system
- ▶ Problems with the System z operating system and applications

This chapter uses the term *zPDT service provider*. This may be an IBM Business Partner or some other zPDT provider that offers service. Not all zPDT users may have such service providers, and some of the information in this chapter may not apply in these cases.

The underlying Linux system, and whatever System z operating system and applications are being used, are not part of the zPDT and are not part of any zPDT support activity. zPDT support includes only the direct zPDT components. As a practical matter, there may be some overlap between Linux issues and zPDT problems, and you may need assistance from your zPDT service provider to isolate the problem.

## 19.1 Problems starting zPDT operation

Many problems are related to devmap errors, and 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.

Problems obtaining a license (from the zPDT token) are typically due to an expired or unactivated token.

Messages about Time Cheat errors indicate a more serious problem. These are typically caused either by (1) moving a token between multiple PCs that do not have their time-of-day clocks closely synchronized, or (2) manipulation of the clock in the PC.

The zPDT system uses several Linux shared storage (virtual memory) areas. These are normally freed when zPDT is ended with an **awsstop** command. A failure or incorrect handling during zPDT startup or operation might result in these shared storage areas not being

released. This can prevent zPDT from being started again. There are Linux commands to individually free shared storage areas, but this requires multiple detailed steps.<sup>1</sup> Rebooting Linux is a primitive but effective way to solve this particular situation.

If you have problems with the token or license consider taking the following actions before seeking help, first resolving any errors detected by these steps:

- ▶ Try the **uimcheck** command.
- ▶ Verify that the local token can be accessed, using the following Linux commands:
 

```
$ lsusb | grep Rain (You see "Rainbow Technologies")
$ ps -ef | grep usbd daemon | grep -v grep (You see safenet_sentinel)
$ ps -ef | grep Sntl | grep -v grep (You see SntlKeysSrvrlnx)
$ netstat -tlp (You see "sntlkeysrvr" in the fourth column)
```
- ▶ Try starting zPDT with a simple devmap, such as this one:
 

```
[system]
memory 1048m
processors 1
3270port 3270
```
- ▶ If a license problem persists, try deleting `/usr/z1090/uim/uimclient.db`. You need *root* authority to do this. Then restart zPDT.
- ▶ Try starting a local browser to "localhost:7002". This should produce a SafeNet browser page.
 

On this page, click on "Keys Information" in the menu list on the right-hand side of the page. This should produce a line showing the serial number of your SENTINEL KEY plus other data. On this line click on the "1" under "Keys#" This should display three lines showing the license numbers installed in your token. These license numbers should be 0xabcd, 0xd98, and 0x8198
- ▶ Try this command; you might need to save the output if further help is needed:
 

```
$ cat /usr/z1090/bin/sntlconfig.xml
```

If you seek help, be prepared to provide the following information:

- ▶ Has your zPDT system worked before the current problem? If so, what has changed?
- ▶ What are the software levels involved?
  - What is the exact zPDT level? A number such as 51.14.01. Use the following Linux command to determine the zPDT level:
 

```
$ rpm -qa | grep 109 (Use "109" not "1090" or "1091")
```
  - What is the Linux level? For example, openSUSE 15.
  - What is the operating system level (and is it an AD-CD system)?
- ▶ Are you running a virtual environment?
- ▶ How much memory is available on the PC (or your virtual machine)?
- ▶ What type of token do you have (a 1090 or 1091)? Is it initialized? Are you certain? (Use Resource Link or your zPDT provider to initialize 1090 tokens. Use IBM Passport Advantage® or your zPDT provider for 1091 tokens, if initialization is needed.)
- ▶ Are you using a remote license server (as opposed to a token in your PC)?
- ▶ Are you trying to use a cloned system for the first time?

---

<sup>1</sup> The Linux **ipcrm** command can be used to remove shared resources that have been orphaned.

## 19.2 Problems during zPDT operation

The zPDT system maintains several logs and traces during operation. The zPDT programs might detect a problem and capture the logs or traces at the time of the problem. You can also capture logs and traces with a **snapdump** command<sup>2</sup>. This command may be used when there is no indication from zPDT that a problem exists, but you detect a problem and might want to work with your zPDT service provider<sup>3</sup> to resolve it.

It is important to remember that this discussion is solely about zPDT operation. The snapdump data is not meaningful for addressing other problems, such as a problem with the System z operating system or System z applications.

A **snapdump** command typically creates a megabyte of data in `/home/ibmsys1/z1090/logs`, contained in various files. You may use **snapdump** as often you want, remembering that each one takes space in the logs directory. Your zPDT service provider might want several, or one, or none of these dumps.

Files in the logs directory created by **snapdump** are retained until you remove them. Over time there may be a buildup of unwanted files in the logs directory. Assuming you are not working on an outstanding zPDT problem, you can simply delete all the files in the logs directory (doing this when zPDT is not running). An easy way to do this is to use the **--clean** option of the **awsstart** command. Again assuming you are not working with a zPDT problem, you might use the **--clean** option every time you perform an **awsstart**. Conversely, do not use the **--clean** option while you are working on a problem; some of the older log and trace files might be wanted at a later time.

The **senderrdata** command is used to package snapdump data into a .tar file, which is typically a little less than a megabyte. This file can be sent to IBM, through your zPDT service provider, or simply kept on your Linux system for potential use later. The **senderrdata** command can manage the FTP operation to IBM. These files (on the receiving system at IBM) are automatically deleted after a few days unless a formal problem (PMR or equivalent) event is opened; this can be done by your zPDT service provider.

Figure 19-1 provides an overview of problem data handling. If snapdump data is sent to IBM (as outlined in the figure), the **senderrdata** option to create a *configuration file* should also be used and the results sent to IBM.<sup>4</sup>

---

<sup>2</sup> The snapdump command is valid only while zPDT is operational.

<sup>3</sup> IBM internal users would communicate through the z1090 forum instead of through a zPDT service provider. Other users, without a defined service provider, might use another zPDT forum.

<sup>4</sup> Later 1090 versions may automatically combine the configuration data with the snapdump data.

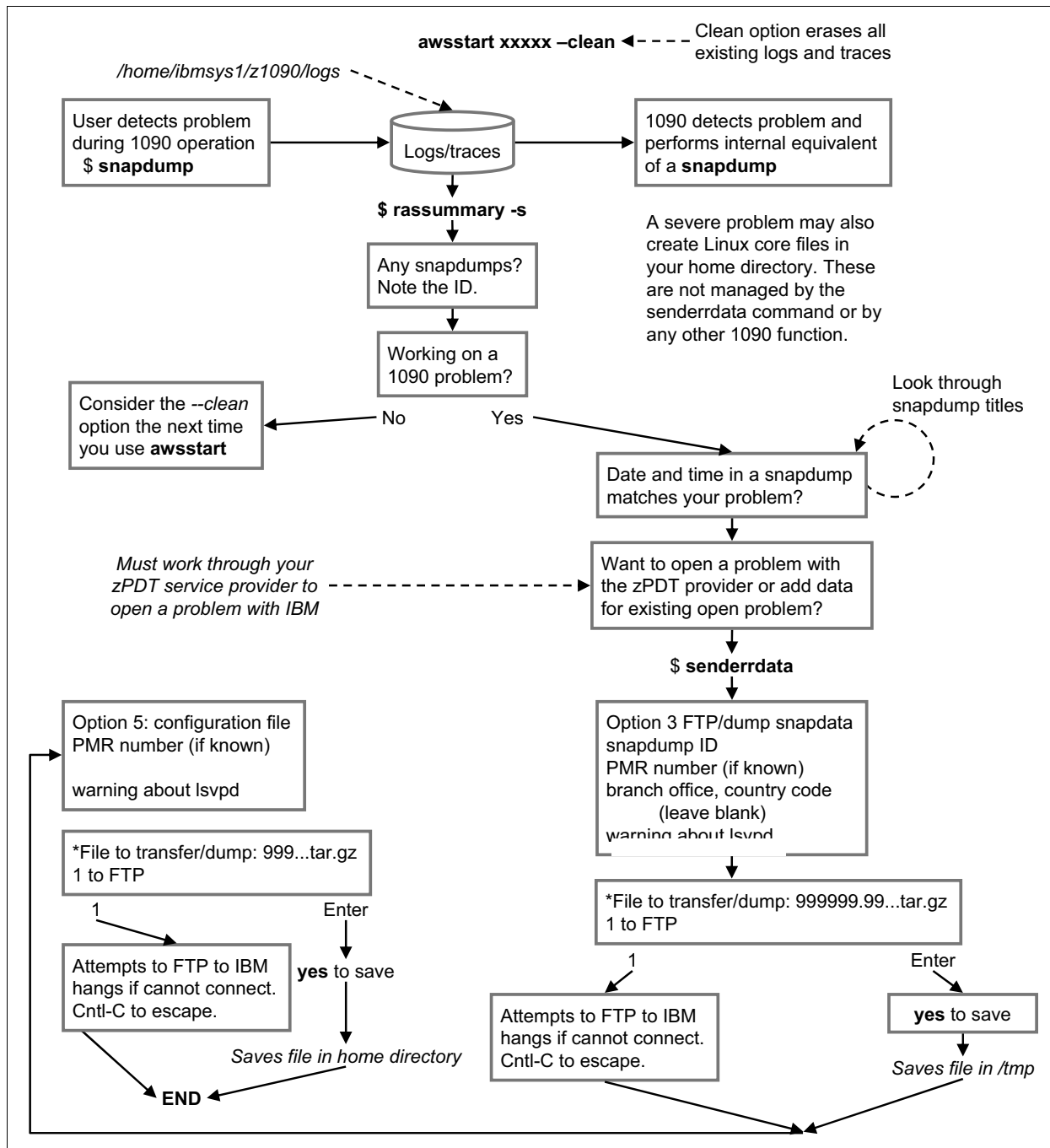


Figure 19-1 Problem data capture and reporting

If a problem incident is opened through your IBM Business Partner, you might be requested to send additional files. In general, after a problem incident is opened, do not delete anything from the logs directory.

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 it controls become *not operational* for the remainder of that zPDT session. The 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).

## 19.3 Core images

Severe problems might cause *core image* files to be created. If these are created by zPDT, they should go into the log subdirectory and be cleaned up with `--clean` option of `awsstart`. If you are actively working on a problem with your zPDT provider, these files may be useful. Otherwise they may be deleted because they can be rather large and might create a disk space problem.

Consistent dumps (“core images”) when zPDT is started can occur if you have a relatively large number of emulated I/O devices (more than 100, for example) and you have not considered memory management adjustments.

The `snapdump` command is used when zPDT is running. If you are unable to start zPDT (with the `awsstart` command) or zPDT ends immediately (before a snapdump can be taken), the problem may have created a core image file. In this case, the core image might help with problem analysis and should be preserved while the problem is under investigation.

## 19.4 Logs

There are many logs that can be references. Some are Linux logs (in `/var/log`) and some are zPDT logs (in `/home/ibmsys1/z1090/logs`, assuming userid `ibmsys1` is used for zPDT operation). The two most useful logs for initially looking at a problem are these:

|                                                            |                             |
|------------------------------------------------------------|-----------------------------|
| <code>/var/log/message</code>                              | <i>(for Linux messages)</i> |
| <code>/home/ibmsys1/z1090/logs/log_console_.....txt</code> | <i>(for zPDT messages)</i>  |

The `log_console` names tend to be long because they include a PID and time/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:

|                                      |                                                 |
|--------------------------------------|-------------------------------------------------|
| <code>\$ alcckd /z/WORK03 -rs</code> | <i>(scan emulated volume for format errors)</i> |
| <code>\$ alcckd /z/WORK03 -rf</code> | <i>(replace bad track with zeros)</i>           |
| <code>\$ alcckd /z/WORK03 -r</code>  | <i>(display volser and size)</i>                |

The `-rs` function scans the emulated volume and verifies that it is in the correct 1090 emulation format.<sup>5</sup> The `-rf` function replaces improperly formatted tracks with a properly formatted track containing zeros. The original contents of the track are lost, but the functionality 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 could 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
```

<sup>5</sup> Only the emulation format is checked. There is no check for data content or operating system metadata (label, VTOC, and such).

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. 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 may be used to verify the format of an awstape file. That is, this command reads the awstape file and verifies that the awstape control blocks are logically correct.

## 19.6 Special problem-related commands

The **senderrdata** command provides a menu of options:

1. rassummary *(uses the Linux less command)*
2. rassummary -s *(for an overview of snapdump incidents)*
3. FTP/dump snapdump data
4. FTP/dump PE directed files *(used only at IBM request)*
5. Create configuration information file
6. Logs Directory maintenance
7. FTP/dump rassummary created files
8. FTP/dump all files in logs directory
9. snapdump

The *dump* options in this menu mean that a .tar file is created, containing the selected files, and the .tar file is saved in /tmp. The dump option (to create a .tar file) is especially useful if the zPDT machine is not connected to the Internet.

The **senderrdata** command also creates a configuration file when you choose option **5**. This file contains the following information such as the version of zPDT, the version of linux, the tokens plugged, memory usage, network configuration, file system information, contents of the z1090 directories, active Linux processes, copy of the last devmap used, and when the *root* password is provided) a copy of the Linux /var/log/messages. This file is useful when debugging events where a linux issue may be impacting the zPDT application.

A number of 1090 commands are intended to be used only at the direction of IBM support 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 are 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 may be helpful. The **vmstat** command is useful and causes very little interference with other processes in Linux.

```
$ vmstat 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
 2 0       0 397600 101936 2398160    0    0   109    52   154   350 14  2 81  3
```

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

Important data from **vmstat** includes the Linux swap rates (si, so); any number here can degrade zPDT performance. If zPDT suffers a page fault, then the whole System z 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
```



## Server Time Protocol (STP)

Server Time Protocol (STP) provides synchronized time-of-day (TOD) clocks among several connected systems. zPDT GA6 (and later) provides this function for use among multiple zPDT systems. The zPDT implementation is not designed for connection to larger System z STP configurations. Figure 20-1 provides an overview of the function.

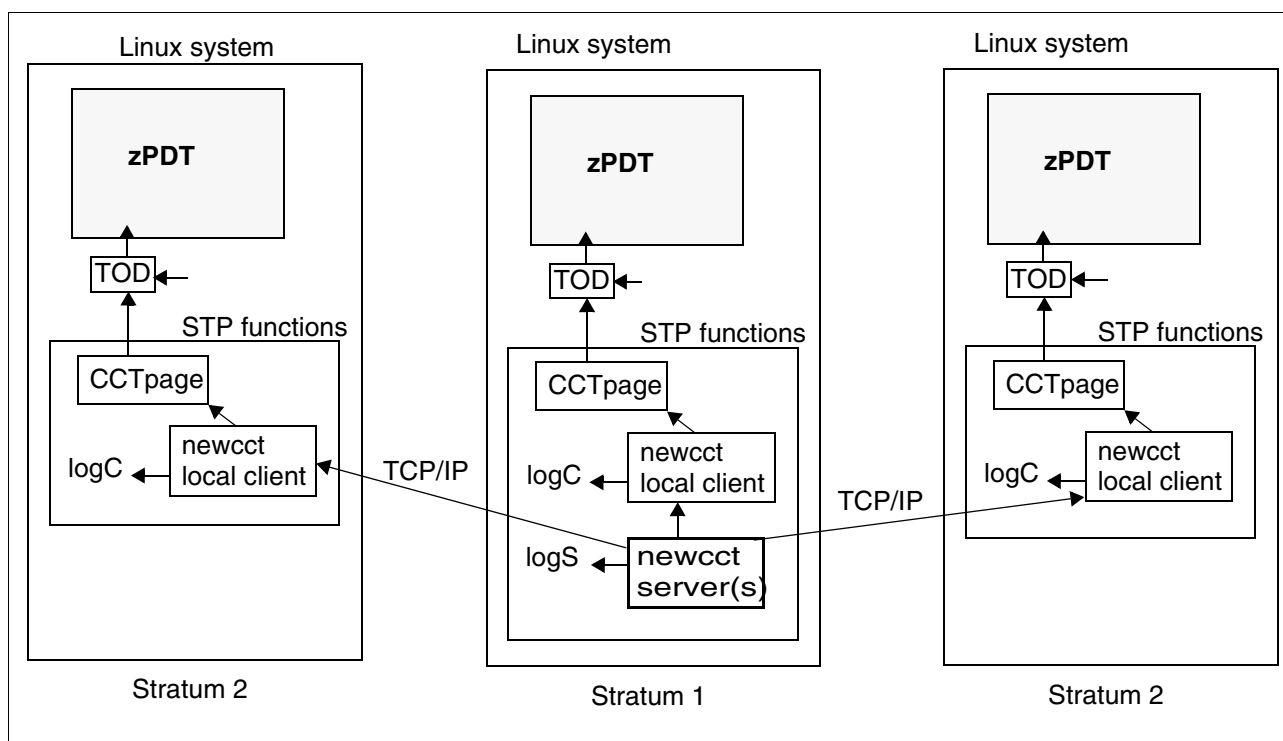


Figure 20-1 STP Overview

STP implementation is considerably more complex than indicated in the figure, but the figure is useful as a starting point in describing the function.

Two terms are important in this description:

- ▶ **Coordinated Timing Network (CTN):** This indicates a collection of servers that are time synchronized. On larger z System 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):** This 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 mixed; this is technically incorrect but the meanings are usually clear. The figure shows three zPDT instances (each in a separate PC) in a CTN. The figure 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 an 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 have not been tested.
- ▶ The stratum 1 server might ultimately obtain its time-of-day from a variety of 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 STP then works from the Linux clocks.
- ▶ The STP server (in the stratum 1 machine) connects to clients. These are *local clients* (in the same Linux as the server) or *remote clients* (in other machines, connected by TCP/IP).
- ▶ STP servers and clients provide log functions. Our implementation (described later) places these in /tmp by default, but this is configurable.
- ▶ The STP executable program is named **newcct**. The same program is used for the server and clients.
- ▶ An STP client maintains a small file named CCTpage, which is in /tmp by default. This contains local timer adjustment data, and is updated as needed by the STP client.
- ▶ Information from CCTpage is used to adjust the time-of-day (TOD) clock that is used by zPDT. Multiple elements contribute to the TOD clock 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 is booted
  - The PC time-stamp counter (TSC), if present
  - The libCCT library package and the ReadCCT routine

The interaction of these elements is complex and is not described in this document.

- ▶ The figure 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. They are started before starting zPDT. In practice, the STP functions are typically started when Linux is booted and are left running while Linux is running.
- ▶ The remote clients use TCP/IP connections to the server. In a larger z System environment, these connections are typically through coupling facility channels, which are very 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 z System CTN.

This chapter describes the practical configuration and operation of STP in a zPDT system. Much more general STP detail is available in *Server Time Protocol Planning Guide*, SG24-7280.

## 20.1 CCT uses

The coordinated time of day provided to multiple z/OS systems can be used for a variety of 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 a parallel sysplex, must have coordinated time-of-day 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.

## 20.2 Configuration

To use the STP function, complete these steps:

1. Create and customize the `/usr/z1090/bin/CCT_data` file, working as *root*. The pattern is given 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 also 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 arbitrary but is a 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 to be 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.
- Remember to place a space before and after the equal sign in this file. Remember that the IP addresses are for the base Linux systems, not for the z/OS systems.

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:

```
ThisNode = 192.168.1.80           (Our second Linux 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; it also automatically adds commands to the Linux **cron** function so that the server is started automatically when Linux is booted or fails. The **stpserverstop** command can later be used to stop the server and remove the automatic cron function. We then start our first Linux, because it contains the STP server. The **stpserverquery** command may be used to determine the state of

your CCT/STP network. After it is added to the cron function, the STP server/clients are automatically started whenever you boot Linux. They do no harm if you are not using them 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 are the names of the Linux systems, as expressed in /etc/HOSTNAME.

The asterisk (\*) denotes the current system which is processing this devmap. In the example, our second system would have 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 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 a **ps aux | grep cct** command for example, 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 time-of-day clock. It affects only clocks that use the CCT interface library routines; in practice, this means zPDT (when a [stp] stanza is present in the devmap) and z/OS (when the CLOCKxx parameter is set).

## 20.3 Additional details

The z/Architecture places tight tolerances on STP clock synchronization and uses the high speed coupling channels as part of the design to achieve the specified tolerances. zPDT, using IP communication, cannot duplicate this environment and 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 these 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 these:

```
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, perhaps, 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 are placed in the Linux directory 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 (remembering that 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/STPClient.monday3PM
```

The fields in the client log are as follows:

|                   |                                                                                                                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>TAI</b>        | International Atomic Time (TAI) is the number of seconds since 1 January 1972, including leap seconds. (The leap second data is not normally available to the zPDT <b>newcct</b> program, and this definition might not be completely accurate.) |
| <b>Offset</b>     | The difference (in microseconds) between the master CCT clock (the stratum 1 server) and the client clock, as calculated by the client.                                                                                                          |
| <b>Dispersion</b> | The uncertainty in the calculated offset value.                                                                                                                                                                                                  |
| <b>Skew</b>       | The difference in the clock stepping rate between the master and the slave.                                                                                                                                                                      |
| <b>Steering</b>   | The rate at which the client clock is being steered in order to reduce the offset toward zero.                                                                                                                                                   |
| <b>RTT</b>        | The minimum measured round-trip delay for a set of CCT sample “ping-pong messages”. More precisely, it is the sum of the minimum forward delay time and minimum backward delay time for a set of CCT sample ping-pong messages,                  |

The “ping-pong” messages mentioned are short message exchanges between a client and server.

The time-of-day provided to z/OS by the CCT/STP function ultimately depends on the time-of-day returned by the effective Linux clock.<sup>1</sup> A user might have a substantially inaccurate time-of-day in the PC BIOS clock that is used to set the Linux clocks when Linux is booted. Another user might have an NTP<sup>2</sup> 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

<sup>1</sup> This is a slightly complex topic because Linux has several clocks; we ignore this detail here.

<sup>2</sup> Network Time Protocol (NTP) references an accurate time-of-day service available over the Internet.

the time data provided by Linux. The STP function keeps multiple zPDT TOD values synchronized, but does not control the accuracy of the time 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 indicates a client failure, but the client immediately restarts itself. We have seen these messages at random intervals on zPDT CCT networks very closely coupled (private 1 GB Ethernet) but have not seen them on other systems connected through the “real” Internet. In our test environment, these transient failures appear to have no effect on z/OS basic sysplex operation. Corresponding log messages on the CCT server include these:

```
Failed to read doorbell msg
Welcome                               (this 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, 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 (using the **awsstart** command) unless you have an STP function active. If you do not have STP active, the zPDT startup will fail. 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 in order 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, increasing the risk of disruptive zPDT STP machine checks. We have performed limited STP testing in a virtualized environment with mixed results and therefore recommend that STP not be used in a virtualized environment.

### 20.3.1 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 additional 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>
```

where:

<active> is the number of leap seconds currently present. (For the first half of 2017 this number is 27.)

<new> is the replacement number of leap seconds. (This is not an additional value, but a complete replacement of the active number. It potentially can be a smaller number than the <active> value, although this has not happened yet.) The difference between <active> and <new> should never be more than one.

<year><month><day><hour><minute><second> is the time at which the <new> value is to be effective.

Two new zPDT commands (issued in a Linux command window) are provided for leap second details:

```
$ d lso                                (display 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
```





# FAQ

The following frequently asked questions (FAQs) and discussions might be helpful to new zPDT users.

## General

**Q:** Are zPDT and zD&T the same thing?

**A:** Partly. The confusion arises because the name “zPDT” is used for two different purposes. In one case, “zPDT” is an IBM offering for Independent Software Vendors (ISVs). In this case “zPDT” and zD&T are not the same, having very different terms and conditions for customers. The other use of “zPDT” is for the emulator that is the base for both the zPDT offering and zD&T. The emulator is essentially the same for the zPDT offering and for zD&T.

**Q:** Is this a multi-user system?

**A:** Yes. Multiple TSO users can connect, in several ways, and use the system in the normal manner. The same applies to z/VM users, CICS users, and so forth.

**Q:** How many users can the system support?

**A:** There is no definitive answer to this. The aws3274 device manager supports 32 connections (which emulate local 3270 devices). There is no specific maximum for TCP/IP (awsosa) connections to z/OS. Practical performance is the primary limitation, not the theoretical connectivity for terminal connections. A given system might do well for one very heavy IBM Db2 or java user, or 10-30 typical TSO users, or 100 web users each having a low transaction rate to a z/OS web server. *The answer to the question depends completely on the nature of the workloads involved.*

**Q:** On my Linux system I routinely login as *root* and then *su* to another userid when necessary. For example, *su ibmsys1* when using zPDT. Is there a problem with this?

**A:** We strongly recommend not doing this. As a general rule, operating as *root* when it is not necessary is simply asking for trouble. Also, you must always respect the difference between *su ibmsys1* and *su - ibmsys1*.<sup>1</sup> The first version (without the “-” character) retains root’s environment while working as ibmsys1. The second version (with the “-” character) switches

<sup>1</sup> Remember that there is nothing special about userid ibmsys1. It is the userid we use in all zPDT examples, but it is not reserved or special in any way.

to ibmsys1's environment and this is the version that should be used if you really must login as *root*.

**Q:** zIIPs are now “free” but zAAPs are not. Why?

**A:** The use of zAAPs is deprecated and do not apply to the z14 level or later.

**Q:** Will adding zIIPs make my system faster, assuming I have sufficient cores?

**A:** Maybe, depending on the workload. However, be aware that with some z/OS parameter combinations (in PARMLIB and WLM), if zIIP(s) exist some workloads will be forced to run *only* on zIIPs even if there are idle CPs. (Recent z/OS AD-CD releases have such parameters.) This is a complex z/OS topic that is not unique to zPDT.

**Q:** The chapter about license servers is a bit confusing. Are hardware tokens being replaced with “software” licenses?

**A:** No, definitely not. The “software-only” licenses are intended for special situations where hardware tokens are not practical, such as with cloud servers. In general, the software-only license servers are more complex to install and manage, and may have additional fees associated with them.

**Q:** Can I install (with **gunzip**) another z/OS volume while zPDT is running?

**A:** In principle, yes. In practice it is not a good idea. Disk use during a **gunzip** of a volume is intense and can generate MIH messages from z/OS.

**Q:** Does the number of zPDT licenses available (assuming a remote license server) equal the number of zPDT z System serial numbers assigned?

**A:** No. See the discussion in “Numbers” on page 166.

**Q:** I purchased an additional zPDT license in order to configure a zIIP. What has changed in zPDT?

**A:** A zIIP no longer needs a separate zPDT license. You might use your extra license to configure an additional CP.

**Q:** Can I install (decrypt) the z/OS AD-CD residence volume from the Dallas 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 AD-CD distribution.

**Q:** Are new z System instructions (as provided with recent new z System machines) present?

**A:** Yes, the emulated instruction set matches the architectural level state for a given release of zPDT. A few instructions dealing with functions not present in a zPDT environment are not available.

**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* at least equal the number of CPs+zIIPs+IFLs. We *strongly recommend* more cores than CPs+zIIPs+IFLs to provide better performance.

**Q:** Will using a zIIP or zAAP increase the performance of my zPDT?

**A:** No, assuming you are replacing a CP with the zIIP or zAAP. These speciality processors operate at the same speed as a “normal” zPDT CP. They are provided to allow developers to verify that their applications use a zIIP/zAAP in the intended manner. Of course, the use of a zIIP or zAAP might allow more parallel operation in your workload, which could increase the performance under zPDT.

**Q:** Do I need to change any z/OS parameters to operate with zPDT?

**A:** In principle, no. In practice, you may need to adjust a few parameters. These are primarily related to performance. For example, the CICS transaction timeout value might need to be

increased for very “heavy” transactions. Obviously, adjustments might be needed for widely different configurations. For example, moving a large Db2 application from a 900 GB LPAR to a 16 GB zPDT laptop might encounter difficulties!

**Q:** I have zPDT and a hardware key. Where can I download z/OS?

**A:** z/OS (or any other IBM z System software) is not part of the base zPDT product. You need to discuss this question with your zPDT provider.

**Q:** Do I need the hardware token to *install* zPDT?

**A:** No, you need it only to *run* zPDT or *install* AD-CD z/OS 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 3390s) do not contain spare cylinders and diagnostic cylinders that may be required for ANALYZE operation. The zPDT command `alckkd -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 z System TOD and timer functions?

**A:** To a large extent, these are approximately as accurate as the timer in the underlying PC. Some interval measurements may have a granularity of about 500 microseconds (plus the z System operating system time needed to manage time-related activities). We suggest you do not depend on very fine timing measurements on zPDT to reflect what timing might be on a larger z System

**Q:** I need to have multiple levels (often more than 3) of z/OS available for testing, although each z/OS is usually idle at any given time. A 1090-L03 seems to be overkill for my modest processing needs and, in any case, is limited to three instances. How can I address this problem?

**A:** The easiest solution is to use z/VM with multiple z/OS guests. This requires some z/VM skills, but these are relatively modest. It probably requires more z System memory than other potential solutions, to avoid excessive z/VM and z/OS paging. See the note in 3.2, “System stanza” on page 34 about older z/OS releases. Memory is important for reasonable performance in such situations. However, you must be sensitive to the architectural level of the zPDT system you are using. For example, the z14 level (zPDT GA8 and GA9) cannot be used with older operating systems that do not support this level.

**Q:** Why do you not provide a definite MIPS or MSU value? This would help us determine how to best use zPDT.

**A:** There are no definite numbers. A MIPS measurement is *very* dependent on the exact workload and your system configuration.

**Q:** Why do some AD-CD 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 AD-CD 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 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 you use. We tend to show SARES1, if mounted, at various addresses.

**Q:** What happens if I remove the hardware key?

**A:** The zPDT functions will stop after a while.

**Q:** You use userid *ibmsys1* throughout all the examples. Is there something special about this userid?

**A:** No. However, you should always use the same Linux userid when running zPDT, and the userid should not be longer than eight characters. (If a *system\_name* statement is used in the devmap, then there is no special limit on the Linux userid name.)

**Q:** Can I use an alternate translation table to convert EBCDIC to ASCII for awspdt output?

**A:** No. However, the translation table was updated for GA9.

**Q:** My z1090 rpm installation failed with an error message about db\_recovery. What now?

**A:** Try the command **rpm --rebuilddb** and then install z1090 again (using the z1090 installer program, and not trying to directly install the z1090 rpm).

**Q:** I am using an emulated printer and this sends output to a Linux file. Does this file remain open for output by zPDT all the while zPDT is running?

**A:** Yes. It is closed if you use **awsmount** to assign a new output file for the printer.

**Q:** Are zPDT commands case-sensitive? Can I issue **ipl** or **IPL**?

**A:** The command names are case-sensitive. They are simply 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*.

**A:** The AD-CD system always starts TCP/IP and associated jobs. How can I delete them?

**Q:** 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 then need to cancel address spaces related to TCPIP, such as INETD1.

**Q:** Can I use RMF?

**A:** Yes, but not all of it is relevant on a zPDT system.

**Q:** How do I IPL the SARES1 volume?

**A:** Assume it is mounted at A9C. The command is **ipl A9C parm 0A9CSA**. Use **S SHUTSA** to shut down the SARES1 system. (It can be mounted at any address defined as a 3390 in the AD-CD 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 is due to memory initialization and management functions being initialized through multiple virtualization paths (VM, SIE, 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.

**Q:** I want to place Db2 buffers in the coupling facilities in my Parallel Sysplex. Also, I want to put the JES2 checkpoint data and RACF data there. How do I do this?

**A:** This is beyond the scope of this document; you need to study the appropriate manuals and/or seek help from a systems programmer with experience in this area.

**Q:** I usually just crash my zPDT system (possibly with the **awsstop** command). Can I continue to do this with the Parallel Sysplex running?

**A:** You can do it, but starting the Parallel Sysplex again may be painful. We strongly suggest you follow the shutdown procedures for a parallel sysplex. In particular, use the **V XCF,xxx,OFFLINE** command to stop additional z/OS members of the sysplex.

**Q:** Why is the z/OS TOD clock (or the z/VM TOD clock) at least 25 seconds different than the base Linux clock?

**A:** The z System operating systems apply leap seconds to the clock; Linux does not do this. The current leap seconds offset, at the time of writing, is 25 seconds. This System z function is included with the zPDT GA6 and later releases. Also, over time while zPDT is running, additional time differences might accumulate.

**Q:** Can I emulate various subcapacity settings and controls?

**A:** No. zPDT does not provide these functions.

**Q:** If I unplug my token how long will zPDT continue to operate?

**A:** The length of operation without a token (or remote license) is variable, but might be up to several minutes.

### **zPDT Configuration, 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 zPDT CP?

**A:** Yes, in theory. z/VM shares the processor with all the guest systems. However, this configuration may 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 be certain you have sufficient PC memory. We suggest 16 GB as a minimum in this case.

**Q:** Are the “free zIIPs” available to both ISV zPDT and zD&T customers?

**A:** Yes.

**Q:** Does IBM need to enable something to allow full operation of the z System internal CRYPTO instructions?

**A:** No, full operation is always enabled.

**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 through A8F. Do I need to define a new awsckd unit in order to add more disk volumes?

**A:** No, you can have up to 256 volumes in one instance of awsckd.

**Q:** All your examples have three-digit emulated device addresses. Is this required?

**A:** No, you may use three- or four-digit addresses. The typical use of only three-digit addresses with the AD-CD z/OS system is a historical accident.

**Q:** Can I use multiple zPDT tokens to obtain more CPs?

**A:** Yes, up to a maximum of 8 CPs in a zPDT instance. In this context zIIPs and IFLs count toward the limit of 8.

**Q:** Can zPDT support older CKD drives, such as 3350s?

**A:** No.

**Q:** Can I use MVS 3.8?

**A:** No. The zPDT system does not support architectures prior to XA and 3380/3390s.

**Q:** Can I configure zPDT to act as, for example, a z800 system?

**A:** No. Each zPDT release matches a particular z System architecture. zPDT GA8 matches z14 systems, for example. There is no 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 something similar?

**A:** We do not know; this was not tested.

**Q:** What are the maximum numbers of CPs, zPDT instances, and I/O devices?

**A:** A maximum of 8 CPs (or combinations of CPs, zIIPs, zAAPs, and IFLs) may be used in a zPDT instance, although your license terms might have a lower limit. A maximum of 15 zPDT instances may exist in a Linux system if you have sufficient licenses. A maximum of 2048 I/O devices may be defined in an instance. *Do not* take these program maximum values as being practical environments. There are other factors (such as available memory, SMP effects, and I/O capability) that limit practical use.

**Q:** Can I move a zPDT token between two PCs?

**A:** Technically, yes,<sup>2</sup> but there is an important issue involved. The latest time-of-day value seen by the underlying PC hardware is stored in the token. If the token then encounters an earlier time, it will fail the operation with a *time cheat* message. If your two PCs have a significant time spread between their hardware time-of-day clocks, you might have problems.

**Q:** You have a whole chapter on STP and CTN (time-of-day functions). Are these typically used with zPDT.

**A:** As far as we know, these are seldom used. They are part of the System z architecture and have been included in zPDT for this reason.

## Base Linux

**Q:** Why do you support only limited Linux releases?

**A:** IBM performs very extensive testing for zPDT. We use Linux releases that are current at the time this testing starts. There are many practical reasons for not changing the Linux level midway in the testing cycles.

**Q:** Are Tumbleweed or Debian supported? If not, why?

**A:** No. Extensive zPDT testing is done for the supported Linux distributions. Resource constraints limit the number of Linux distributions that can be supported.

**Q:** I have several Linux windows open while running zPDT. I can enter 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. 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 very large to avoid worrying about it?

**A:** As far as zPDT is concerned, you could do this. However, it is possible that other Linux applications might accept the very large value and attempt to use unreasonable amounts of shared virtual memory, resulting in excessive paging or system failure.

**Q:** Why is zPDT placed in `/usr`? This directory should be only for basic Linux components.

**A:** You are correct, but it is difficult for zPDT to change now.

**Q:** Can I run as *root* when installing and using 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 userid (such as *ibmsys1*).

---

<sup>2</sup> You should, of course, observe the terms and conditions of your zPDT license agreement.

**Q:** Does zPDT operate in kernel mode? In suid mode?

**A:** Kernel mode is not used, but a few modules operate in suid mode. However, these details might change with future zPDT releases.

**Q:** Can I use a dual boot method to place Windows and Linux on the same machine?

**A:** Yes, provided you have sufficient disk space. The primary challenge might be to prevent Linux or Windows updates from overwriting the dual boot functions.

**Q:** Can I routinely migrate to the next Linux releases when they become available?

**A:** Maybe. There is no unique zPDT tie to a particular release, although the Linux release should be the same or later than the zPDT “build” level. However, it is possible that the 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. Consult your zPDT provider.

As a general answer, you are probably safe going to a new minor distribution release. For example, going from RHEL 7.3 to RHEL 7.4 should be safe. We recommend not jumping to a new major distribution, such as from RHEL 7.x to RHEL 8.x, that is not listed as a “Base” level in “zPDT releases” on page 24. Our experience is that minor zPDT installation adjustments are often needed for new major Linux release levels.

**Q:** Is there any Linux maintenance 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 zPDT (this is /home/ibmsys1 in our examples) and delete any *core* files. These can be quite large and are usually unwanted unless you are actively debugging a problem. Note that *core* files in the zPDT subdirectories should be investigated.

We have no specific recommendations about online updates to your Linux base system. In earlier years, we avoided these because of various problems that were introduced through the updates. More recently, some members of the zPDT development team have been routinely doing online updates of their Linux (when zPDT is not running) without experiencing problems related to the updates.

**Q:** Can I use compressed disks or compressed disk files or sparse files to reduce the amount of disk space needed by z/OS volumes?

**A:** zPDT does not support any form of compressed disks/files or Linux sparse files. If you have a compressed disk scheme that is completely transparent to zPDT you could try it.

## **zPDT operation**

### **Tapes, SCSI drives, awstape**

**Q:** I have a SCSI tape drive. I want to use it directly for Linux functions (not connected with 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. You might need some detective work to find it.

**Q:** Can I use a SCSI DLT tape drive?

**A:** It should work (if it supports the *SSC-3 SCSI Command Set for Sequential Devices*), although this 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 we suggest you do not use 4 mm drives. These have proven to be poorly suited for emulated IBM S/390 work.

**Q:** Can awstape files from P/390 or R/390 systems be used with the zPDT offering?

**A:** In general, yes. There is a restriction that the P/390 or R/390 awstape file cannot be read beyond the last valid logical data record. The older awstape files do not contain the proper indicators after the last logical data record. (However, awstape files created by zPDT work correctly in this situation.) This situation is typically encountered when using “tape map” programs that attempt to read everything on a tape, without obeying the normal EOVS/EOF records or double tape marks normally 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. Notice there is a double dash before the wtm option.

## PC hardware, cores, channels, memory

**Q:** Most of the documentation is about large laptops or servers. I have a typical desktop PC. Can I use zPDT with it?

**A:** Probably, assuming Linux works properly with the display, DVD drive, power, and LAN interfaces on your desktop. You should have *at least* 8 GB of memory (more is better). However, the only formal support is for the machines described in this document. IBM simply cannot undertake the extensive testing that would be needed to qualify the vast variety of PCs that exist.

**Q:** Could I use the Linux **taskset** command to bind a Linux core to a zPDT CP? Would this make the hardware caches more efficient?

**A:** The zPDT developers have not tried this. Remember (1) **taskset** 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, and (2) there are many zPDT processes for I/O operation and JIT functions. In a given situation, especially with sufficient cores available, you might improve zPDT performance slightly but this would probably take considerable experimentation and be sensitive to a particular workload. Attempting to automate such tuning for all zPDT installations and workloads could become very complex and have the potential to degrade performance instead of enhance it.

**Q:** Does zPDT reserve PC processor cores for z System execution? Does it partition PC memory in some way to create z System memory?

**A:** The answer is *no* to both questions. (See a prior question about the Linux **taskset** command.) A running zPDT system consists of many processes and threads under Linux; these are dispatched in the normal Linux manner and reference Linux virtual memory 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 simply results in a slower system because the single processor must handle all z System CP operations plus all the other processes for I/O and other Linux details. (The one-core machine is an exception to the recommendation that there should be at least one more core than the number of CPs.)

**Q:** I am short on USB ports. Can I use a USB extender for the token connection?

**A:** Do not use an *unpowered* USB port extender; it might render your zPDT token unusable. A powered USB port extender should work correctly.

**Q:** Can I use a USB disk drive for emulated z System data?

**A:** Yes, assuming the base Linux recognizes and supports the drive in the normal manner. It

might offer slightly less performance than the internal PC disk drives, but this 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. zPDT does not care about these settings; it simply 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 and this enhances performance.

**Q:** Is there an adapter for parallel channels?

**A:** There are currently no hardware channel adapters for zPDT systems.

**Q:** I already have ESCON adapters from previous products for my Intel base PC. Can I use these?

**A:** No.

**Q:** Can a solid-state disk drive be used (in the PC) instead of a traditional hard disk?

**A:** Yes, they are very 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. Additional SCSI drives on a larger server, especially with multiple SCSI adapters, *might* help. We would be interested in any documented experiences in this area.

**Q:** I am configured to use the emulated crypto adapter but RMF says there is no adapter. What am I doing wrong?

**A:** RMF uses a CHSC command to obtain statistics from a crypto adapter. zPDT does not currently support this CHSC command. Use other means to verify that the crypto adapter is working as expected.

## **3270 sessions, emulators**

**Q:** How many “local” 3270 sessions can I use with zPDT?

**A:** At the time of writing, the limit is 32 sessions with the aws3274 device manager. There is no particular limit on the number of sessions through z/OS TCPIP.

**Q:** Can I use 3270 sessions on other PCs? Your example has all the sessions on the zPDT machine.

**A:** Yes, of course. Simply point your 3270 emulator (on your external PC) to the Linux IP address and port 3270 (if you are using port 3270 for aws3274, of course). We suggest using a relatively modern 3270 emulator, such as recent versions of PCOMM or x3270. Older, “free” 3270 emulators have created problems during some of the zPDT test cycles.

**Q:** Do the int3270port and intASCIIport interfaces provide the same functions as the equivalent functions on an HMC attached to a larger z System?

**A:** This is the intended operation, although the operational characteristics might differ.

**Q:** Can I use my “brand X” TN3270e client?

**A:** Maybe, but do not base any error reports to IBM on it. Not all TN3270e clients are the same and there can be significant differences in the handling of error and recovery situations.

**Q:** Does zPDT handle 3270 nulls correctly?

**A:** This is not a function of zPDT; it is a function of the 3270 emulator and, to some extent, the application involved. Relevant functions for x3270 can be found in **Options** → **Toggles** → **Blank Fill**. The 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 zPDT. Every time I start Personal Communications it wants to print something. How can I stop this?

**A:** This is a well-known issue, and is not related to zPDT. Personal Communications stores user profiles in .ws files (such as `bill.ws`, for example). Find the .ws profile you are using and add the following lines at a reasonable place in the profile:

```
[LT]
IgnoreWCCStartPrint=Y
```

**Q:** I want to use PCOMM instead of x3270. Is this acceptable? Can you include it with the zPDT package?

**A:** PCOMM is part of a separate IBM product. We cannot include it as part of the zPDT package.

## OSA and LANs

**Q:** The `--interface` parameter for `awsosa` is confusing. How should I use it?

**A:** Read the material in Chapter 7, “LANs” on page 115. The `--interface` parameter was introduced because recent Linux distributions have changed the way LAN interfaces are named and a more general method of specifying a LAN interface to `awsosa` was needed.

**Q:** Will my devmap from a previous zPDT release work with a new zPDT releases?

**A:** Probably. An important issue is with path names for OSA interfaces. Previous zPDT releases considered only LAN interfaces that were not *down* when assigning path names. The current release considers all detected LAN interfaces and this may result in a different path name for OSA.

**Q:** Can I have multiple tap (tunnel) interfaces, such as `tap0`, `tap1`, and so forth?

**A:** Yes. A total of eight OSA devices of any type may be defined. The tunnel interfaces will typically have CHPID numbers `A0`, `A1`, `A2`, and so forth. The tap devices may be defined (as seen by the `find_io` command), 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 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 these.

**A:** A full discussion is beyond the scope of this document. We recommend you use the QDIO (also known as OSD) interface, in which case the unit addresses are not needed. For non-QDIO TCP/IP, you need to ensure that the unit addresses are 0 and 1. (This is required by the default OAT used by OSA.) You need to ensure that unit address `FE` is used only for OSA/SF when using the default OAT.<sup>3</sup> You need to remember that the default unit address is the same as the low-order two 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.

---

<sup>3</sup> Starting with z/OS 2.3, OSA/SF is not being used.

**Q:** Does zPDT support thin interrupts?

**A:** Yes, for OSA device emulation, but not for CF communication. (This function is properly known as the Adapter Interrupt Facility.)

**Q:** Can I filter IP traffic before it is sent to my emulated OSA-Express interface? This reduces the overhead involved in rejecting packets not addressed to my system.

**A:** In OSD (QDIO) mode, there is some automatic filtering. In OSE (non-QDIO) mode, you can customize the OAT with your IP address. If this is done, the OSA interface will pass only packets intended for this IP address.<sup>4</sup> 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 NAT functions on the base Linux, then most of the filtering is done at that level.

**Q:** Is OSN operation (CDLC) provided with current zPDT OSA emulation?

**A:** No.

**Q:** Should I use 1492 or 1500 as the maximum packet size (MTU) when using awsosa?

**A:** We use 1492. The details are beyond the scope of this document. (As best we can tell, the z System communication routines automatically adjust this number down if necessary. Thus it probably does not matter whether you specify 1492 or 1500.)

**Q:** Does current 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 we believe the first OSA address for a TCP/IP stack must be an even number. You would need to use 400-402, skip 403, then use 404-406, skip 407, and so forth. (This statement might not be correct for z/VM.)

**Q:** Do the current zPDT OSA offload functions work? Do they accomplish anything on an emulated system?

**A:** The Linux-based zPDT OSA implementation does not use offload functions at this time. In some cases (with current Linux kernels) you might need to force Linux to disable offloading.

**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. We tried an Xterasys Gigabit PC Card (98-012084-585). We also informally tried several older IBM 10/100 EtherJet cards. However, none of these are “supported” zPDT options and we have doubts about more recent Linux support for such adapters.

**Q:** Can I use IP aliasing in Linux while using zPDT?

**A:** Yes, but the alias addresses are not relevant to zPDT and are not displayed by the `find_io` command.

**Q:** I have multiple Ethernet adapters, each on a different subnet. Response is very slow and I get multiple responses to pings. Is there a problem using multiple adapters?

**A:** In principle, no. However, multiple interfaces on different subnets should not be connected to the same VLAN. This 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. We suggest not using multiple LAN adapters unless you have the necessary networking skills available.

---

<sup>4</sup> However, the former program (OSA/SF) used to edit OATs is no longer available (starting with z/OS 2.3) and the replacement program (Queryinfo) is not supported by zPDT.

**Q:** I have an error message about GVRP when I try to use a VLAN/VSWITCH in z/VM. Is this 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 ICC provided?

**A:** No. However, the AWS3274 device manager provides approximately the same service.

**Q:** I sometimes want to change Linux TCP/IP between DHCP and a static IP address. Can I do this while zPDT is running? I am changing only Linux parameters, not OSA parameters.

**A:** This is not supported, not tested, and probably will not work correctly. We suggest you do not change Linux LAN definitions while 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 can also use it for aws3274 clients if you find a client (and Linux host) that supports IPv6. As a practical matter, there has been minimal zPDT testing of IPv6.

**Q:** You say that Ethernet SNA operation is not *supported*. Might it work?

**A:** Yes, it *might*. It has not been tested and IBM will not respond to problems using it.

## Non-QDIO OSA

The awsosa device manager can emulate QDIO or non-QDIO operation. The mode is selected by the *pathtype* parameter in the devmap. Type OSD specifies QDIO operation and type OSE specifies non-QDIO operation. Non-QDIO operation is often noted as *LCS operation* or *3172 operation*, although these descriptions are not exactly correct. Non-QDIO operation can involve TCP/IP or SNA (or both), although SNA usage is not tested or supported with zPDT. We strongly suggest you always use OSD mode unless you have a specific reason for using OSE mode.

IBM does not support Ethernet SNA operation for zPDT. This means that problems or defects are not addressed by IBM if you attempt to use SNA operation over Ethernet. However, we are aware that some zPDT users have worked with Ethernet SNA successfully. If you attempt this, consider the following information:

- ▶ The default 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.
- ▶ While SNA performance may be acceptable for simple testing, it is unlikely to be acceptable for heavier usage.
- ▶ A given PC ethernet adapter can be used for OSA-Express emulation in either QDIO or non-QDIO<sup>1</sup> mode, but not both. The selection of QDIO or non-QDIO is made in the devmap definitions; the awsosa device manager is used in both cases.
- ▶ The OSA/SF tool is no longer available with z/OS and the partial replacement (**QUERYINFO**) is not functional for z/PDT. The effect is that the default OSE (non-QDIO) configuration (the “OATs”) of an emulated OSA cannot be easily changed.
- ▶ We strongly recommend that OSD (QDIO) be used unless you have a very 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:

<sup>1</sup> Non-QDIO mode emulates an older IBM 3172 control unit or LAN Channel Station, (LCS). This mode is not recommended. It is required for SNA connectivity, although there is no zPDT support for SNA usage. Non-QDIO operation is also known as OSE operation.

## 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 CTC or OSA definition for addresses E20 and E21.<sup>2</sup> Different addresses can be used, of course, but they must match the IODF in your z/OS system. The HOME address and ROUTE statements in the example are just examples, of course. The GATEWAY statements could be used instead of the ROUTE statements. The --unitadd parameter is used in the devmap because the default OSA unit addresses<sup>3</sup> would be 20 and 21 (using the *two* low-order digits of the device number) and we want unit addresses 0 and 1.<sup>4</sup>

Which mode should you use for OSA connectivity? QDIO mode has many benefits for TCP/IP use on a larger z System; it reduces the z System workload and provides automatic sharing of the adapter across multiple LPARs. These considerations do not fully apply to a zPDT system. The following points are relevant:

- ▶ QDIO operation offloads some processing from the zPDT CP to the Linux processor. The offloading is not as much as on a larger machine, but it helps. It also reduces the number of z System instructions 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; it does not handle SNA.
- ▶ QDIO can provide VSWITCH, IPv6, and Enterprise Extension connections.
- ▶ Non-QDIO operation can mix TCP/IP and SNA (or handle just SNA or just TCP/IP). However, SNA operation with zPDT is not supported.
- ▶ Suitable non-QDIO (LCS) devices have been defined in z/OS AD-CD systems. (These are the CTCs starting at address E20.)

Other than these points, there is no practical difference between using QDIO or non-QDIO on a zPDT system. In particular, the user at a TN3270 TSO session cannot detect the difference. Normal TCP/IP functions, such as FTP and Telnet, do not detect any differences.

<sup>2</sup> LAN operation in LCS mode can use CTC definitions in the z/OS IODF. This is a carryover from earlier LAN implementations.

<sup>3</sup> This unit address is the (emulated) hardware address within the (emulated) OSA control unit. It is not the device number ("address" in common terminology).

<sup>4</sup> The default OAT used by OSA requires unit addresses 0 and 1 for TCP/IP when in OSE mode.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *zPDT 2020 Sysplex Extensions*, SG24-8386-02
- ▶ *IBM System z Personal Development Tool Messages and Codes*, SG24-8103-00
- ▶ *Installing Linux for z Systems on zPDT: A Short Cookbook*, SG24-8330-00
- ▶ *Communications Server for z/OS V1R9 TCP/IP Implementation Volume1: Base Functions, Connectivity, and Routing*, SG24-7532.
- ▶ *Introduction to the New Mainframe: z/VM Basics*, SG24-7316
- ▶ *Server Time Protocol Planning Guide*, SG24-7280
- ▶ *IBM Rational Development and Test Environment for System z Installation and Sample Configuration Guide*, SC14-7281
- ▶ *IBM Rational Development and Test Environment for System z Activation Guide*, SC27-6630

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Other References

The following documents may help with background information and specific details concerning zPDT and z/OS operation in this environment.

- ▶ *Cheryl Watson's Tuning Letter*, multiple editions:  
<http://www.watsonwalker.com>
- ▶ *Running Mainframe z on Distributed Platforms*, by Kenneth Barrett and Stephen Norris of CA (formerly Computer Associates). The book is available from various places (ISBN-13 978-1-4302-6430-9) or as an electronic deliverable (ISBN-13 978-1-4302-6431-6).

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](https://ibm.com/services)

# Index

## Symbols

.bashrc, changes 100  
/etc/profile.local file 27  
/etc/sudo file 81, 261–262  
/etc/sysconfig/network 135  
/etc/sysctl.conf 99  
/opt/IBM/LDK/rules.ini 164  
/usr/z1090/bin 19, 101  
/usr/z1090/bin/awsCCT file 86  
/usr/z1090/bin/librarybuild 265  
/usr/z1090/bin/sntlconfig.xml 153  
/VM 6.4 AD-CD OSA 140  
\$SPRT1 command 236

## Numerics

1090 messages 266  
1090 token 89  
1091 token 89  
1403 printer 233  
3215 terminal 191  
3270 display, larger 222  
3270 emulator 16  
3270 interfaces 116  
3270 nulls 354  
3270 sessions 353  
3270 sessions, starting 110  
3270port parameter 116  
3270port statement 35  
32-bit version of zPDT 27  
3350 drives 349  
3390 volume, moving 226  
3390 volumes, additional 112  
3390, emulated, ANALYZE 347  
4-mm tape drive 351

## A

absolute address 66  
acc command 187  
accelerator function (crypto coprocessor) 8  
activation, token 167, 169  
AD system, device addresses 108  
Adaptec ASC-29320ALP U320 287  
Adapter Interrupt Facility 355  
AD-CD system, installation 107  
AD-CD z/OS 2.1 357  
addresses for z/OS 347  
Adjunct-processor stanza 38  
ADRDSU 228, 287  
adstop command 30  
AHCI disk mode 353  
aksusbd service 156  
alcckd command 26, 40, 52  
alcckd command, diagnosis 333

alcckd, spare volume 261  
alcfba command 54  
alias addresses 118  
alias function, Linux 260  
aliasing, IP 355  
Allowed Site Address 163  
ap\_create command 55  
ap\_destroy command 55  
ap\_query command 55  
ap\_von and ap\_voff commands 55  
ap\_vpd command 56  
ap\_zeroize command 56  
APAR OA50068 256  
Architecture levels 9  
ASCII and EBCDIC, awsrdr 47  
ASCII, underlying host 8  
asn\_lx\_reuse operand 255  
asn\_lx\_reuse parameter 255  
asynchronous data movers 8  
attn command 56  
Authorized User List 163  
aws\_bashrc command 57, 99  
aws\_config command 99  
aws\_findlinuxtape command 57, 284  
aws\_sysctl command 57  
aws\_tapelnit command 57, 191, 274  
aws\_tapelnsnp command 58  
aws2scsi command 288  
aws3215 11  
aws3215 device manager 49  
aws3270 device manager 115  
aws3274 11  
aws3274 device manager 42, 116  
awsckd 11  
awsckd device manager 40  
awsckd, number units 349  
awsckd, spare devices 261  
awsckmap command 58  
awscmd 11  
awscmd device manager 48, 207, 274  
awsctc 11  
awsctc device manager 50  
awsfba 11  
awsfba device manager 41  
awsin command 191  
awslog command 334  
awsmount command 21, 33, 44, 47, 234, 272  
awsmount command 49  
awsmount, new DASD 262  
awsmount, SCSI tapes 283  
awsmount, with awsoma 50  
awsoma 11  
awsoma device manager 49  
awsosa 11  
awsosa device manager 30, 45

- awsOSA performance 133
- awsprt 11
- awsprt command 30
- awsprt device manager 47
- awsprt usage 234
- AWSPRT, translation table 348
- awsrdr 11
- awsrdr device manager 46
- awsscsi 11
- awsscsi device manager 49, 282–283
- awsstart 110
- awsstart command 23, 60, 350
- awsstart window 260
- awsstart, problems 331
- awsstat command 30
- awsstop command 62, 111, 348
- awstape 11
- awstape device manager 44
- awstape scenarios 272
- awstape utilities 279
- awstape, compaction 44
- awstape, format 106
- awstape, P390 352

## B

- Backup servers 159
- base Linux 7
- base machine 7
- BCPii functions 8
- BIOS 254
- Block counts, 3490 285
- block size, SCSI 287
- Blocked Site Address 163
- BLP processing 209
- bonded Ethernet interfaces 17
- bonding, LAN 117
- BPXTCAFF program 296
- browse command 187

## C

- card2tape 279
- card2tape command 62, 279
- card2txt command 63
- CCT\_data file 339
- CCWs for awstape 275
- CCWs, tape 30
- CEX4C level 28
- CEX5S adapter 173
- CFCC level 27
- CFCC Level 18 28
- CFCC level 23 30
- Channel command 4B 275
- Channel command 54 275
- channel connections 270
- Channel measurement blocks 271
- Channel Measurement Facility 9
- channel/device measurement 30
- Cheryl Watson's Tuning Letters 246
- CHSC command 353

- CHSC commands 9
- CKD versioning 265
- ckdPrint command 63, 334
- client and server details 151
- client, migration 289
- clientconfig command 63, 152, 154, 164, 169, 262–263
- clientconfig\_authority command 64, 169, 263
- clientconfig\_cli command 64, 153–154, 169
- clientconfig\_cli line command 155
- Cloning zPDT 159
- CMS, Conversational Monitor System 178
- COD z/OS system 71
- command
  - acc 187
  - alckd 52, 333
  - alcfba 54
  - aws2scsi 288
  - awsckmap 58
  - awslog 334
  - awsmount 33
  - awsstart 23, 60
  - awsstop 62, 111
  - browse 187
  - card2tape 62, 279
  - card2txt 63
  - ckdPrint 63, 334
  - cpu 65
  - directxa 187
  - discard 187
  - diskmap 187
  - doOSAcmd 334
  - dreg 334
  - dshrmem 334
  - fbaPrint 67
  - filelist 187
  - find\_io 45, 138
  - format 187
  - ind 187
  - interrupt 69
  - ipl 70, 110
  - ipl\_dvd 70
  - link 187
  - loadparm 72
  - managelogs 73
  - memld 73
  - msgInfo 74
  - mt 351
  - oprmsg 74
  - peek 187
  - printlog 334
  - printrace 334
  - purge system prt all 187
  - purge system rdr all 187
  - q accessed 186
  - q all 187
  - q alloc all 186
  - q alloc map 186
  - q da all 186
  - q disk 186
  - q links 120 186

- q n 187
- q pf 186
- q prt 187
- q stor 186
- q system 186
- query 76
- rassummary 77, 331
- ready 78
- receive 230
- rel 187
- restart 78
- scsi2tape 79, 288
- senderrdata 81, 331
- snapdump 83
- st (store) 83
- start 84
- stop 85
- storestatus 85
- sys\_reset 86
- tape2file 87, 279
- tape2scsi 87
- tape2tape 88
- tapeCheck 88, 334
- tapeheck 279
- tapePrint 89
- token 89
- top 335
- txt2card 90
- vmlink 187
- vmstat 335
- xmit 228
- z1090instcheck 93, 348
- z1090ver 94
- command statement (in devmap) 35
- compatibility mode, disk 353
- compressed disks 351
- compressing PARMLIB 246
- console log files 258
- console, emulation operation 14
- Copying DASD volumes 255
- core image files 333
- core images, startup 333
- core, terminology 7
- Coupling Facility code 27–28
- cp command 256
- CP, definition 7
- CPs, CPUs, threads, tokens 265
- CPs, maximum number 10
- cpu command 65
- cpuopt statement 35, 255
- crontab entries 261
- crontab, for OSA 136
- crypto adapter, RMF 353
- CRYPTO instructions 349
- crypto instructions 27
- cryptographic adapter 38
- cryptographic adapter functions 27–28
- CSFKGUP use 321
- CTN logs 338
- Customized Offerings Driver (COD) 247

CZAM 247, 255

## D

- Dallas site, maintenance 219
- DASDVOL, RACF class 293
- dataset encryption 317
- Date Extension 169
- db\_recovery, rpm error 348
- DB2 buffers 348
- Debian 350
- def\_reserved\_size, SCSI 287
- desktop PC 352
- det command 187
- device addresses, for AD 108
- device manager
  - aws3215 49
  - aws3274 42
  - awscmd 48
  - awsfba 41
  - awsprt 47
  - awsrdr 46
  - awsscsi 49
- device map 7
- device map (devmap) 33
- DEVICE name 119
- device statements 40
- devices, maximum 267
- devmap 7, 23
- devmap, environmental variable 37
- devmap, example 21
- devmap, for AD 109
- devmap, introduction 21
- devmap, z/VM 175
- devmapvm devmap 175
- DFLTCC instruction 9
- DHCP 133
- DHCP and local router 132
- DHCP client 133
- DHCP-assigned addresses 152
- directory structure 22
- directxa command 184, 187
- DIRMAINT 185
- Disabled wait delay 219
- disabled wait state 19 255
- disabled waits, z/OS 247
- discard command 187
- disk (hard disk) changes 159
- disk changes 159
- disk space, planning 20
- diskmap command 187
- display\_gen2\_acclog command 67, 157
- docker usage 325
- domain statement 38
- doOSAcmd command 334
- dreg command 334
- dshrmem command 334
- DTCPARMS file, z/VM 188
- dual boot 351
- dynamic changes to devmap 261
- dynamic changes to the DASD environment 41

## E

- EAV allocation 53
- EBCDIC and ASCII, awsrdr 47
- EBCDIC, usage 8
- eDMosa module 263
- EIO, CHPID type. 40
- emulated volumes, file system 22
- Emulex Corporation Zephyr-X 286
- ENABLE ALL command 177
- environmental variable, devmap 37
- ESCON adapters, other 353
- ESCON connection, zBox 270
- Ethernet adapters, multiple 355
- Ethernet ports, PCMCIA 355
- Ethernet SNA 140
- ethtool command 136
- ethtool commands 134, 136
- expand statement 34
- expect rpm 97
- extended measurement word 271
- Extreme configurations 15

## F

- Factory Reset option 164
- fbaPrint command 67
- FCB functions 47
- Fibre open system FBA 41
- FICON 8
- FICON connection, zBox 270
- filelist command 187
- find\_io command 45, 67, 138
- firewall functions 96
- firewall, Linux 132
- firewall-cmd 125, 128
- firewall-cmd command 131
- Firewalls 159
- flash storage 9
- force userid 187
- format command 187
- four-digit addresses 349
- fstab options 296
- Fujitsu M2488E 286

## G

- Gemalto N.V., product names 4
- Gen1 License server 156
- Gen1 tokens, definition 5
- Gen2 tokens, definition 5
- gen2\_init 169
- gen2\_init command 30, 68
- Generic VLAN Registration Protocol 46
- Generic VLAN Registration Protocol 117
- gunzip utility 107
- GVRP for OSA 46
- GVRP, for OSA 356
- gzip command 256
- gzip, compression 106

## H

- hardware console 30
- hardware key 347
- Hardware Management Console 9
- HCD, usage 230
- hckd2ckd command 68–69, 295
- Health Checker 221
- hfba2fba command 68, 295
- hibernate state, z/VM 253
- hibernation, PC 269
- hipersockets 9
- HMC 9
- host Linux 7
- hot reader, JES2 46
- htape2tape command 295
- Hyper-Threading 254

## I

- IBM 2107 control unit 30
- IBM 3592-E05 TS1120 286
- IBM LTO-3 3580 286
- ibmsys1 userid 107, 348
- ibmsys1, userid 13–14, 97
- ibmsys2 and ibmsys3, userids 13
- IBUSER password 111
- ICKDSF ANALYZE 347
- ICKDSF job 112
- IEBCOPY problems 219
- IFAPRDxx, PARMLIB member 218
- IFASMF DL job 246
- IFL processor 34
- include function 37
- include statements 27
- ind command 187
- inet in z/OS 135
- installation, rpm problem 348
- Installer options 98
- installer program 98
- instance name 30
- instances, zPDT 13
- instruction set 346
- int3270port statement 35
- intasciiport statement 35
- interface parameter 354
- interface parameter 354
- Interfaces, LAN 68
- interrupt command 69
- INVALID REGISTER NUMBER 342
- IO DF changes 230
- IO DF entry, 3592 282
- IP address, changing 356
- IP traffic, filtering 355
- ipl command 14, 35, 70, 110
- IPL in ESA390 mode 255
- IPL operation 110
- IPL sequence, speed 348
- ipl statement 35
- ipl\_dvd command 70, 173
- IPLing in ESA/390 mode 30

- iptables 129
- iptables command 136
- iptables commands 125, 136
- IPv6 usage 356
- IQPPRM00, PARMLIB member 218
- iso fonts 102
- ISPF, starting 238
- IZU=xx parameter 221

## J

- Java and WAS startup 224
- JES2 checkpoint data 348
- JES2 print 236
- JES2 setup 235
- Jumbo frames 119
- jumbo frames 134

## K

- kernal.shmmax parameter 267
- kernel mode 351
- kernel.msgmni 268
- kernel.sem parameters 268
- kernel.shmall parameter 267
- kernel.shmall value 100
- kernel.shmmax 350
- kernel.shmmni 268
- kernel.shmmni parameter 267

## L

- LAN adapter 20
- LAN interfaces 68
- LAN setup 132
- laptop usage 349
- LDK licenses 5
- ldk\_server\_config 169
- ldk\_server\_config command 71
- Leap seconds 342
- lease date 80
- Lexmark printers 233
- libncurses5 library 96
- librarybuild, information 265
- License expiration notification 160
- License renewal 165
- License search order 160
- license server 27–28
- License server, Gen1 156
- License server, Gen2 156
- License servers, virtual environment 326
- link command 187
- LINK parameter 119
- Linux CKD formatting 53
- Linux command window 260
- Linux firewall 132
- Linux for System z 173
- Linux LAN definitions 356
- Linux monitoring 335
- Linux releases, new 351
- Linux userids 13

- list-directed IPL 8
- listVtoc command 27, 72
- load\_daemon.sh 165
- loadparm command 72
- loadserv 165
- local 3270 sessions 353
- local token 151
- local zPDT system 149
- log files, STP 339
- log files, zpdt 258
- logical channel subsystems 9
- logs, Linux 333
- logs, zPDT 333
- logs/traces 331
- logstreams, deleting 246
- LPAR name 30
- LPARs 9
- lpr 235
- lso (leap second information block) 343
- LUnicode, aws3274 device manager 42
- LX and ASN REUSE facility 255

## M

- MAC address 68
- mail command 160
- MAINT userid 178
- Maintenance for AD-CD z/OS 219
- man files, installation 98
- managelogs command 73
- 39
- manager stanzas, general 39
- maxlength parameter, awstape 44
- Measurement Facility, CPU 8
- media, software 106
- memld command 73
- memory statement 34
- Memory, PC 19
- memory, PC, size 353
- message function 37
- message numbers, 1090 266
- Message Security Assist 11, 27
- message security assist enhancements 11
- message statements 27
- messages, 1090 266
- MIDAW operation 30
- MIDAWs 8
- migrating 3390 volumes 226
- MIH (Missing Interrupt Handler) 259
- MIH problems 12
- Minidisks and files 179
- minimum z/OS storage 217
- MIPS 347
- MIPS (million instructions per second) 15
- MNTCCWS option 275
- modprobe 287
- mount command, Linux 257
- mount\_dvd command 73
- MSA9 functions 9
- msgInfo command 74
- msgmax and msgmnb settings 100

- mt command 351
- mt package for Linux 288
- MTU for jumbo frames 135
- MTU size 134, 355
- multiple CPUs 13
- Multiple I/O paths 9
- Multiple tokens, updating 168
- Multithreading 9
- multi-user system 345
- MVS 3.8 349
- MVS console, lost 237

## N

- name statements 39
- NAS disks 17
- NAT functions 152
- NAT router 133
- net.core.rmem\_max parameter 133
- Network File System 15
- network-attached storage 17
- new release 103
- newcct 338
- Information Technology Company (ITC) 270
- NFS client 257
- NFS use 256
- NFS use with zPDT 140
- NOGVRP parameter 356
- non-QDIO 357
- nt3270port function 36

## O

- OAT defaults 45
- OAT, default 354
- OAT, filtering 355
- OAT, multiple instances 197
- operating systems, multiple 112
- oprmsg command 74
- oprmsg command, alias 260
- oprmsg usage 237
- oprmsg\_file 35–36
- oprmsg\_file statement 258
- ordering information 95
- OSA (Open Systems Adapter) 7
- OSA address conflicts 140
- OSA interfaces 354
- OSA notes 267
- OSA QDIO with IP6 356
- OSA QDIO, multiple 355
- OSA/SF 354, 357
- OSA-Express functions 45
- OSA-Express interface, filtering 355
- OSA-Express, limits 46
- OSA-Express2 offload functions 355
- OSA-ICC 356
- OSE CHPID type 46
- OSN operation 355
- out of memory, Linux 261
- oversize parameter, x3270 222

## P

- P/390, AWSTAPE 352
- packet size, TCP/IP 355
- paging, disks 353
- paging, with z/VM 187
- PAGING63 parameter 175
- Parallel Access to Volumes (PAV) 9
- parallel channel 353
- password, IBMUSER 111
- patch file 98
- path assignments 68
- PATH for emulated devices 40
- path parameter 40, 117
- pathtype parameter 40
- PAV 8
- PAV (parallel access to volumes) 41
- PC Card 355
- PC printer 233
- PCMCIA card, Ethernet 355
- PCOMM, Linux 354
- pdsUtil command 26, 75
- peek command 186–187
- performance 14
- performance, z/VM 15
- Perl, installation 97
- Personal Communications product 354
- PING command 137
- ping-pong messages 341
- PKCS#11 8, 307
- port 1023 245
- port 3088 267
- port 3270 267
- port 3990 267, 294
- port 7002 163
- port 9450, for token 267
- port 9451 267
- port number, list 267
- PRG009 ABEND 253
- printer output 350
- printing 233
- printlog command 334
- printtrace command 334
- processor, terminology 7
- processors statement 34
- PROFILE TCPIP, z/VM 188
- purge system prt all command 187
- purge system rdr all command 187
- PUT tapes 219

## Q

- q accessed command 186
- q all command 187
- q alloc all command 186
- q alloc map command 186
- q da all command 186
- q disk command 186
- q links 120 command 186
- Q Logic Corporation ISP2432 287
- q n command 187

- q pf command 186
- q prt command 187
- q rdr 187
- q stor command 186
- q system command 186
- QDIO interface, multiple 355
- QDIO operation, advantages 358
- QDIO or non-QDIO operation 357
- QDIO setup 119
- qpci 334
- query command 76
- query\_license command 77, 157, 169
- QUERYINFO command 46

## R

- RACF® data 348
- RAID5 usage 259
- RANDOM option 162
- RAS, basic 3
- RAS, comments 1
- RAS, general statement 3
- rassummary command 77, 331
- Rational License Server 35, 171
- rdrlst 187
- rdrlst command 186
- rdtserver statement 35
- read-only DASD 256
- read-only volumes 256
- ready command 78, 272
- receive command 230
- Redbooks website 359
  - Contact us xiv
- rel command 187
- releases, new 103
- reliability, scalability, or serviceability 3
- remote operation 267
- renewal, token license 167
- request\_license 170
- request\_license command 78
- request\_license program 156
- RESERVE and RELEASE 41
- RESOLVER and java 221
- resolver, z/OS 130
- Resource Link 103, 169
- restart command 78
- RMF 348
- root mode 7
- root userid 350
- router personal 132
- routers 140
- rsync command 256

## S

- S071 ABEND 222
- SAD program 224
- SafeNet 4
- SafeNet browser page 330
- SafeNet driver programs 18
- SafeNet module restarts 164

- safenet-sentinel 156, 163
- SALIPL command 176
- SAP processors 34
- SARES1 ipl 348
- Scenario 2 122
- Scenario 3 124
- Scenario 4 126
- Scenario 5 128
- SCSI adapters 283
- SCSI block size 287
- SCSI def\_reserved\_size 287
- SCSI DLT tape drive 351
- SCSI tape 351
- SCSI tape drives 283
- SCSI tape, block counts 285
- scsi2tape 288
- scsi2tape command 79, 288
- SecureUpdate\_authority command 80, 170, 262
- SecureUpdateUtility 27
- SecureUpdateUtility command 80, 170, 262
- security exposure 262
- security, integrity, and RAS concepts 3
- security, license server 163
- SEL protection 132
- selection menu, 3270 205
- senderrdata command 81, 331, 334
- Sentinel Admin Control Center 154
- sentinel\_shk\_server restart 269
- serial number, changes 159
- server, migration 289
- serverconfig command 81, 170
- serverconfig\_cli command 82, 170
- service command, Linux 269
- set pf12 retrieve 187
- settod command 82, 260
- shared option 41
- Shared read-only volumes 256
- shell script, printing 234
- SHK tokens 5
- shk\_usb restart 269
- shk-server 98
- shmall value 30, 100
- shmmax value 30, 99
- SHUTDOWN command, z/VM 177
- shutdown, z/OS and 1090 111
- SIMD (Vector Facility) 29
- SLES-10 SP1 (for System z) 296
- SMB use with zPDT 140
- SMF recording 246
- SMP/E 219
- smpppd rpm 36, 97
- SMS setup 319
- SNA operation 11, 357–358
- snapdump 331
- snapdump command 83
- sntlconfig.xml 153
- sntl-sud 98
- software media 106
- software-only license 2
- solid-state disk drive 353

- sparse Linu files 351
- spin loop timeouts 222
- spinloop problems 255
- SPINLOOPS, cause 12
- st (store) command 83
- stand-alone dump 224
- Stand-alone dump output dataset 225
- stanza 34, 39
- start command 84
- START name 119
- stop and start commands 27
- stop command 85
- storage for z/OS 217
- STORAGE volumes 223
- storestatus command 85
- storestop command 30
- STP and CTN 350
- STP function 86
- stpserverquery command 86, 339
- stpserverstart command 86, 339
- stpserverstop command 86, 339
- stratum levels 338
- subcapacity settings 349
- subchannels, maximum 12
- Suspend and Hibernation 269
- SVC dumps, space 223
- sys\_reset command 86
- SYS1.LOGREC full 236
- sysctl parameter 133
- Sysplex Extensions, SG24-8386-02 359
- SYSTCPT DD statement 137
  - 34
- System Recovery Boost 9
- system stanza, devmap 34
- System Time Protocol (STP) 29
- system timer protocol (STP) function 39
- System z 196 processor 27
- System z Personal Development Tool 7
- System z software 105
- system\_name 35–36

## T

- tap (tunnel) interfaces 354
- tape CCWs 30
- tape2file 279
- tape2file command 87, 279
- tape2scsi 288
- tape2scsi command 87
- tape2tape 279
- tape2tape command 88
- tapeCheck 279
- tapeCheck command 88, 334
- tapeheck command 279
- tapePrint 279
- tapePrint command 89
- taskset command 352
- TCP/IP port 3990 291
- TCP/IP stacks, multiple 354
- TCP/IP, Linux 356
- TCP/IP, starting z/OS 348

- tcpmaint, z/VM use 188
- telnet connection 245
- telnet session 135
- thin interrupts 355
- time cheat 350
- time cheat message 260
- timer functions, accuracy 347
- TKE system 307
- TN3270E client, other 353
- TN3270e connections 116
- TOD clock 349
- TOD steering 8
- token command 26, 28, 89, 158, 170
- token port number 267
- Token Ring 356
- token types 148
- token updates 167
- token, dates 260
- token, moving 350
- top command 335
- TRACERTE command 137
- Transport channel commands 9
- Transport Mode I/O 8
- TRL entry 119
- TSO commands, NETSTAT 137
- Tumbleweed 350
- tunnel connection, usage 135
- tunnel environment, setup 45
- txt2card command 90

## U

- UCS functions 47
- UDX, cryptographic adapters 8
- UIM configuration 154
- UIM ContactServer 153
- UIM serial number 150
- UIM server 155
- uimcheck command 90, 155, 170
- uimclient.db, deleting 330
- uimd service 27
- uimreset command 90, 151, 155, 170
- uimreset -l command 150
- uimreset -r command 164
- uimserverstart command 90, 170
- ulimit command 100
- ulimit -m and -v 268
- underlying host 7
- Unique Identifier Manager, UIM 148
- unit address, OSA 354
- unitadd parameter 40
- Universal Time 97
- Unsupported Function, 3270 43, 111
- update\_license 170
- update\_license command 91
- update\_license program 156
- uPDT Enterprise Server 167
- USB 3 ports 27
- USB disk drive 352
- USB extender 352
- USB port expander 17

userid, Linux 13  
users, maximum 345

## V

V XCF,xxx,OFFLINE 348  
VATLSTxx member 223  
Vector Facility 29  
VIPA functions 355  
VLAN networks 152  
VLAN, usage 355  
VLAN/VSWITCH in z/VM 356  
VMAC support 138, 201  
VMCOM1 volume 175  
vmlink command 187  
vmstat command 335  
VMWare Player 326  
VMWare products 326  
VNC use 267  
vsftp, selection 97  
vswitch support 138, 201  
VTAM commands 137  
VTAMLST ATCCON00 119

## W

wait state 19 255  
workloads, concurrent PC 12

## X

x3270 101  
x3270 command 37  
x3270 fonts 102  
x3270 oversize parameter 222  
x3270 scripting 272  
x3270, startup 110  
x3270, to z/OS TCP/IP 135  
x3270if 272  
XEDIT editor 182  
xmit command 228

## Z

z System serial number 148  
z/OS 106  
z/OS CP and memory display 220  
z/OS operator console, lost 237  
z/OS TCP/IP 116  
z/OS, download 347  
z/OS, older releases 255  
z/TPF 189  
z/VM 106  
z/VM 6.4 174  
z/VM cold start 176  
z/VM directory 183  
z/VM IPL and logon 175  
z/VM PRG009 ABEND 253  
z/VM TCP adjustments 188  
z/VM, performance 15  
z/VSE 106, 173  
z1090 rpm 98

Z1090\_ADCD\_install 107  
Z1090\_ADCD\_install command 91, 107  
Z1090\_removal command 93  
Z1090\_token\_update command 92, 170, 262  
z1090instcheck command 27, 93, 101, 263, 348  
z1090term command 28, 36, 93  
z1090ver command 94  
Z1091\_ADCD\_install command 91, 107  
Z1091\_token\_update command 92, 262  
z1091ver command 94  
z196 processor 27  
zAAP and zIIP, creation 34  
zAAP speciality engine, z13 9  
zAAP speciality processor, z13 29  
ZARCH\_ONLY=NO 255  
zBX functions 8  
zData Appliance 270  
zEDC (DEFLATE) 217  
zEDC emulation 30, 217  
zIIP or zAAP, performance 346  
zKey appliance 166  
ZOSSERV.XMIT file 292  
zPDT build information 265  
zPDT instance name 30  
zPDT log files 258  
ZPDT\_EXP\_EMAIL environmental variable 160  
ZPDTMSRV module 292  
zPDTSecureUpdate command 170  
zpdtSecureUpdate command 80, 94, 262  
zVM\_CouplingFacility 255  
zVM\_CouplingFacility function 35





Redbooks

# IBM zPDT Reference and Guide

(1.5" spine)  
1.5" <-> 1.998"

789 <-> 1051 pages



Redbooks

# IBM zPDT Reference and Guide: System z Personal Development Tool

(1.0" spine)  
0.875" <-> 1.498"  
460 <-> 788 pages



Redbooks

# IBM zPDT Reference and Guide: System z Personal Development Tool

(0.5" spine)  
0.475" <-> 0.873"  
250 <-> 459 pages



Redbooks

# IBM zPDT Reference and Guide

(0.2" spine)  
0.17" <-> 0.473"  
90 <-> 249 pages

(0.1" spine)  
0.1" <-> 0.169"  
53 <-> 89 pages



**Redbooks**

# IBM zPDT Reference and Guide

(2.5" spine)  
2.5" <-> nnn.n"

1315<-> nnnn pages



**Redbooks**

# IBM zPDT Reference and Guide

(2.0" spine)  
2.0" <-> 2.498"

1052 <-> 1314 pages





# IBM zPDT Guide and Reference

## System z Personal Development Tool



**Redbooks®**

### IBM System z Personal Development Tool

### Full IBM z/OS usage

### Linux base

This IBM Redbooks publication provides both introductory information and technical details for the IBM System z Personal Development Tool (zPDT), which produces a small System z environment suitable for application development. zPDT is a PC Linux application. When zPDT is installed (on Linux), normal System z Operating Systems (such as IBM z/OS) may be run on it. zPDT provides the basic System z architecture and provides emulated IBM 3390 disk drives, 3270 interfaces, OSA interfaces, and so forth.

This current document merges four separate previous Redbooks publications into this single book. The primary reason for this merger is to provide simpler zPDT documentation usage when viewing or searching the documentation onscreen.

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

This book provides the primary documentation for zPDT and includes basic system overview, installation, operation, z/OS distribution, FAQs.

### INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

### BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)