

z/OS  
3.2

*MVS Planning: APPC/MVS Management*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 247](#).

This edition applies to IBM® z/OS® 3.2 (5655-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2025-09-30

© **Copyright International Business Machines Corporation 1991, 2025.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.....</b>	<b>ix</b>
<b>Tables.....</b>	<b>xv</b>
<b>About this information.....</b>	<b>xvii</b>
Who should use this information.....	xvii
How to use this information.....	xvii
Where to find more information.....	xviii
<b>How to provide feedback to IBM.....</b>	<b>xix</b>
<b>Summary of changes.....</b>	<b>xxi</b>
Summary of changes for z/OS 3.2.....	xxi
Summary of changes for z/OS 3.1.....	xxi
<b>Part 1. Introduction.....</b>	<b>1</b>
Chapter 1. Introduction to APPC/MVS.....	3
APPC Overview.....	3
How APPC Relates to SNA, LU 6.2, VTAM, and CPI-C.....	3
APPC Concepts and Commonly Used Terms.....	5
Programming Terms.....	5
Network Terms.....	7
What is APPC/MVS?.....	9
Programming Support for APPC/MVS Callable Services.....	10
z/OS System Support.....	13
Overview of an APPC/MVS Outbound Request.....	14
Overview of an APPC/MVS Inbound Request.....	15
Chapter 2. Planning Overview.....	19
Levels of Connections.....	19
Physical Connections.....	20
Program Connections.....	20
Logical Connections and APPC/MVS Management.....	21
APPC Management Tasks.....	21
System-Wide APPC Connections.....	21
<b>Part 2. Program management.....</b>	<b>25</b>
Chapter 3. Scheduling Transaction Programs.....	27
Overview of Transaction Scheduling.....	27
Scheduling Characteristics of the APPC/MVS Transaction Scheduler.....	28
Using the APPC/MVS Transaction Scheduler.....	29
Classes of Transaction Initiators.....	29
DISPLAY Command.....	30
TP Schedule Types.....	30
Multi-Trans Processing.....	30
SMF Accounting of Multi-Trans Resources.....	32
Security for Multi-Trans TPs.....	32

SYSOUT Processing for Multi-Trans TPs.....	32
Assigning Multi-Trans TPs to their own Class.....	32
Establishing a Multi-Trans TP that is Always Available.....	33
Logging Transaction Program Processing.....	33
The TP Message Log.....	34
Chapter 4. Defining Scheduling Characteristics with ASCHPMxx.....	43
ASCHPMxx Parmlib Member.....	43
Changing Values.....	43
Using Default Values.....	44
Planning Specific Values.....	45
Defining a Class — CLASSADD Statement.....	45
Example of defining a class.....	45
Modifying a Class — CLASSADD Statement.....	47
Example of modifying a class.....	48
Deleting a Class — CLASSDEL.....	49
Example of deleting a class.....	49
Defining Default Options — OPTIONS.....	50
Example of Defining a Default Class.....	50
Defining Default Scheduler Options — TPDEFAULT.....	50
Example of Defining Scheduling Defaults.....	51
Examples ssing ASCHPMxx Parmlib members.....	51
Tracking Changes in Scheduling Definitions.....	53
Chapter 5. Controlling the Execution of Transaction Programs.....	55
Determining Scheduling Characteristics.....	55
Defining the VSAM Key Sequenced Data Sets (KSDS).....	56
Determining How Many Files to Define.....	56
Determining the Size of Each File.....	57
Creating a TP Profile.....	59
TP Profile Key.....	59
Program Attributes Section.....	60
Transaction Scheduler Section.....	61
Summary of TP Profile Keywords.....	64
Creating Side Information.....	65
Example of Side Information.....	66
Summary of Side Information Keywords.....	67
Defining TP Profiles and Side Information Early.....	67
Specific Scheduler JCL Information for TP Profiles.....	67
SYSOUT Recommendations.....	68
JCL Size Restrictions.....	68
Unsupported Statements and Restrictions.....	68
Chapter 6. Using the APPC/MVS Administration Utility.....	73
Utility Commands.....	73
TP Profile Commands.....	73
Side Information Commands.....	74
Database Token Commands.....	74
Syntax Requirements.....	74
Syntax Rules.....	75
Invoking the APPC/MVS Administration Utility.....	76
Invoking the APPC/MVS Administration Utility from a Batch Job.....	77
Invoking the APPC/MVS Administration Utility from an Application Program.....	78
Invoking the APPC/MVS Administration Utility from a REXX Program.....	79
Restrictions on Invoking the APPC/MVS Administration Utility.....	80
Return Codes.....	80
Examples.....	80

Chapter 7. Using the APPC/MVS Administration Dialog.....	83
Overview of the Dialog.....	83
TP Profile Administration.....	83
Side Information Administration.....	84
Database Token Administration.....	84
How to Use the Dialog.....	84
Using a Command Line.....	84
Using PF Keys.....	85
Using Input Fields.....	85
Receiving Messages and Getting Help.....	86
Installing the Dialog.....	87
Installing the Dialog under Application Manager.....	87
Installing the Dialog from ISPF.....	89
Customizing the Dialog.....	90
<b>Part 3. Session management.....</b>	<b>93</b>
Chapter 8. Planning Sessions.....	95
Determining the Number of Local LUs.....	95
Defining the System Base LU.....	95
Naming LUs.....	96
Using Network-Qualified Names Support.....	96
Assigning a VTAM Generic Resource Name to APPC/MVS LUs.....	98
Setting Up a Session for APPC/MVS.....	103
Defining a Local LU on MVS.....	104
Defining an APPC Logon Mode.....	104
Defining the Local LU to VTAM.....	105
Customizing Sessions for APPC/MVS.....	110
Defining Additional Logon Mode Entries.....	111
Specifying a Logon Mode for a Conversation.....	111
Using APPC/MVS Protected Conversations Support.....	111
Chapter 9. Controlling Configuration through APPCPMxx.....	121
APPCPMxx Parmlib Member.....	121
Changing Values.....	121
Default Values.....	121
Planning Specific Values.....	122
Adding a Local LU — LUADD Statement.....	122
Example of Adding LUs.....	123
Modifying a Local LU — LUADD Statement.....	123
Examples of Modifying an LU.....	124
Deleting a Local LU — LUDEL Statement.....	125
Examples of Deleting an LU.....	126
Specifying a VSAM KSDS for Side Information — SIDEINFO Statement.....	126
Examples Using APPCPMxx Parmlib Members.....	127
Initial APPC Setup.....	127
Anticipated Modifications.....	127
Deletions.....	128
Tracking Changes in the Configuration.....	129
Keeping a Hardcopy Log.....	129
Viewing the Current Configuration.....	129
<b>Part 4. Security management.....</b>	<b>131</b>
Chapter 10. Setting up Network Security.....	133
APPC/MVS Security Requirements.....	133

Giving the APPC and ASCH Started Procedures Access to Resources.....	133
Why Security for APPC?.....	133
An APPC Application Example.....	134
Planning for APPC Security.....	134
Determining the Application's Security Type.....	135
LU Security Mechanisms.....	136
Conversation Security Mechanisms.....	137
LU Security: Protecting APPC/MVS Logical Units.....	137
Coding Security Keywords on the VTAM APPL Statement.....	137
Defining LU-to-LU Access Authority with RACF APPCLU Profiles.....	139
Controlling the Use of VTAM ACBs.....	143
Conversation Security: Protecting APPC/MVS TPs.....	144
Establishing a Security Environment for Inbound TPs on MVS.....	145
Controlling User Access to LUs.....	145
Controlling User Access from LUs.....	148
Controlling User Access to TP Profiles and Side Information on MVS.....	148
Controlling Ability to Collect API Trace Data.....	154
Obtaining SYSOUT and Account Information from RACF User Profiles.....	156
Using Persistent Verification (PV).....	157
Diagnosing Security Problems.....	159
Maintaining MVS Passwords in an APPC Environment.....	159
What is the SIGNON/Change Password TP?.....	160
How to Create Partner LU Communication for the SIGNON/Change Password TP.....	160
Using Sample Programs to Maintain User Passwords on a Partner LU.....	165
Diagnosing Problems when Using the Password Maintenance Sample Programs.....	168
How to Install the Sample Programs that Maintain Passwords.....	178
Encrypting Data and Security Information.....	180
Summary.....	180

## **Part 5. System management..... 183**

Chapter 11. Operating APPC/MVS.....	185
Starting the APPC and ASCH Address Spaces.....	185
Restarting APPC/MVS.....	186
Displaying Information about APPC/MVS Work.....	186
Dynamically Changing the APPC/MVS Environment.....	187
Changing Parmlib Specifications through the SET Command.....	187
Changing LU Characteristics through VTAM Commands.....	188
Stopping APPC/MVS Work.....	188
Deactivating a Transaction Program through its TP Profile.....	188
Stopping an Initiator Address Space with the STOP Command.....	189
Stopping a Class of Transaction Programs with the SET Command.....	189
Stopping One or More LUs with the SET or VARY Command.....	190
Stopping a TP or APPC/MVS Address Space with the CANCEL Command.....	192
Stopping VTAM with the HALT Command.....	193
Summary of Methods of Stopping APPC/MVS Work.....	194
Tracking Changes to the APPC/MVS Configuration and Workload.....	196
Displaying TP Status.....	196
Displaying UR Status.....	197
Displaying Server Status.....	197
Displaying LU Status.....	198
Displaying Scheduling Status.....	198
Managing Use of the APPC/MVS API Trace Facility.....	204
Planning for the Use of API Trace Data Sets.....	204
Restoring API Tracing Capability.....	204
Recovering from APPC problems.....	205
Recovery for the APPC Address Space.....	205

Recovery for the APPC/MVS Transaction Scheduler (ASCH) address space.....	206
Chapter 12. APPC/MVS Measurement and Tuning.....	209
Managing APPC Work in the System.....	209
Considering APPC/MVS Storage Requirements.....	210
Changing the Maximum for the System Workload.....	210
Monitoring APPC Performance.....	210
Using SMF to Audit APPC Work.....	210
Using RMF Reports.....	212
Using the DISPLAY Operator Command.....	213
Improving Performance Through Program Design and Administration.....	213
Making Efficient Use of Callable Services.....	213
Avoiding Certain JCL Keywords.....	213
Using the Multi-Trans Schedule Type.....	214
Defining Classes and Response Time Goals.....	214
Putting Multi-Trans TPs in their Own Class.....	215
Associating TPs and LUs with the Appropriate Level.....	215
Limiting Use of the TP Message Log.....	216
Improving Performance through System Changes.....	216
Controlling Buffer Limit Size.....	216
Minimizing Use of APPC Component Trace.....	218
Controlling SMF Type 33 Recording for APPC.....	219
Improving Network Performance.....	219
Maximum number of APPC active conversations.....	220
<b>Part 6. Installation checklists.....</b>	<b>223</b>
Chapter 13. Installing an APPC Application.....	225
Installing a TP that Initiates an Outbound Request.....	225
Installing a TP that Responds to an Inbound Request.....	228
<b>Appendix A. Character Sets.....</b>	<b>233</b>
<b>Appendix B. Coding the APPCLOG Utility.....</b>	<b>237</b>
Parameters.....	237
Examples of using the APPCLOG Utility.....	239
Sample output.....	239
APPCLOG Formatted Dump.....	239
APPCLOG Analysis Dump.....	242
<b>Appendix C. Accessibility.....</b>	<b>245</b>
<b>Notices.....</b>	<b>247</b>
Terms and conditions for product documentation.....	248
IBM Online Privacy Statement.....	249
Policy for unsupported hardware.....	249
Minimum supported hardware.....	249
Programming Interface Information.....	250
Trademarks.....	250





---

# Figures

1. Network Communications between LUs and Users.....	3
2. An SNA Network for Communications between Different Systems.....	4
3. CPI Communications Program Scenario.....	5
4. A Session between Two LUs.....	7
5. A Conversation between Two TPs.....	8
6. Parallel Sessions between LUs.....	8
7. Different Types of Sessions between Two LUs.....	9
8. Types of APPC/MVS Callable Services.....	10
9. Using TP Profiles and Side Information to Find a Partner TP.....	12
10. Using Side Information in Client/Server Communications.....	13
11. APPC/MVS Communications Services (Outbound).....	15
12. APPC/MVS Communication Services (Inbound).....	16
13. Levels of Connections.....	19
14. APPC Connectivity Options.....	20
15. System-Wide APPC Connections (Part 1 of 2).....	23
16. System-Wide APPC Connections (Part 2 of 2).....	24
17. Transaction Program Routing.....	28
18. Initialization and Termination in Multi-trans Processing.....	31
19. Example of Adding a TP Profile.....	34
20. Message Wrapping in a Multi-Trans TP Message Log.....	38
21. Messages in a Cumulative TP Message Log for a Multi-Trans TP.....	39
22. TP Profile Parameters.....	39
23. ASCHPMxx Parameters.....	40

24. Invoking the Write Log Routine.....	40
25. ASCHPM00.....	44
26. ASCHPM1A.....	46
27. DISPLAY command output.....	47
28. DISPLAY command output.....	48
29. ASCHPM1M.....	48
30. ASCHPM1D.....	49
31. DISPLAY command output.....	49
32. ASCHPM2A.....	50
33. ASCHPM3A.....	51
34. DISPLAY command output.....	51
35. ASCHPM1S.....	52
36. ASCHPM2M.....	52
37. ASCHPM1M.....	52
38. ASCHPM1D.....	53
39. ASCHPM2D.....	53
40. ASCHPM3D.....	53
41. SET command LIST option output.....	54
42. DISPLAY command output.....	54
43. Relationship of files to LUs.....	56
44. Combinations of Partner LUs, TPs, and Logon Modes.....	57
45. Side Information Estimate.....	57
46. TP Levels in a File.....	58
47. TP Profile Estimate.....	58
48. TP Profile Key.....	59

49. Mapping an SNA TPNAME into an Administration Utility TPNAME.....	60
50. Program Attributes Section.....	60
51. APPC/MVS Transaction Scheduler Section.....	61
52. Side Information Key.....	65
53. Side Information Data.....	65
54. Mapping an SNA TPNAME into an Administration Utility TPNAME.....	66
55. Example of Side Information.....	67
56. Example of JCL to Invoke ATBSDFMU Using SYSIN Data.....	77
57. Example of JCL to Invoke ATBSDFMU Using a SYSIN Data Set.....	77
58. Sample IKJTSOxx PARMLIB member.....	88
59. Sample ICQASE00 ICF Member.....	89
60. Sample ISPF Panel Definition for APPC/MVS Selections.....	90
61. Sample ISPF Selection Panel with APPC/MVS Options.....	90
62. APPC/MVS Configuration with a Mix of Allocate Requests to Specific and Generic LU Names.....	99
63. Example of an LUADD Statement.....	104
64. Example logon mode (APPCPLM).....	104
65. Example logon mode (MVSAPPC).....	105
66. Application Definition to VTAM.....	108
67. COBOL example of an ATBALC2 call using a logon mode name.....	109
68. COBOL example of a CMINIT call using a symbolic destination name.....	110
69. Side Information for USR3NEWS.....	110
70. Overriding Session Defaults.....	110
71. APPCPM1A.....	123
72. APPCPM2A.....	124
73. APPCPM2M.....	124

74. APPCPM3M.....	125
75. DISPLAY command output.....	125
76. APPCPM1D.....	126
77. APPCPM2D.....	126
78. APPCPM1A.....	127
79. APPCPM3S.....	128
80. APPCPM1S.....	128
81. APPCPM1D.....	128
82. APPCPM2D.....	128
83. APPCPM3D.....	128
84. SET command LIST option output.....	129
85. DISPLAY command output.....	130
86. Sample APPC/MVS Conversation.....	134
87. Passing Security_PGM Information on an Allocate Request.....	136
88. Sending Security Information through VTAM.....	138
89. Security for LU01 and LU02.....	144
90. Setting Security Environment from the RACF Profile.....	145
91. TP Profiles and Side Information.....	149
92. Protecting the Generic User ID.....	153
93. Permitting Update Access to a Multi-trans TP.....	153
94. Signing a User On to APPC/MVS (Unexpired Password).....	160
95. Signing a User On to APPC/MVS (Changing an Expired Password).....	161
96. Changing an Expired Password.....	167
97. APPC Error Notification Panel.....	168
98. DOS Error Notification Panel.....	169

99. APPCPM1A.....	187
100. APPCPM2A.....	187
101. Example of Deactivating a TP.....	188
102. Sample DISPLAY Output.....	189
103. ASCHPM4D.....	190
104. APPCPM1D.....	190
105. Sample DISPLAY Output.....	192
106. Sample DISPLAY Output for System SY1 on Network USIBMY0.....	199
107. Sample DISPLAY Output for System SY2 on Network USIBMZ0.....	200
108. Sample DISPLAY Output.....	201
109. Sample DISPLAY Output.....	202
110. Sample DISPLAY Output.....	203
111. Sample DISPLAY Output.....	203
112. Example of a TP Profile for No Account Tailoring.....	212
113. Example of a TP Profile for Account Tailoring.....	212
114. Example of a RACF User Profile for Account Tailoring.....	212
115. Response Time in an APPC/MVS Environment.....	215
116. Example.....	225
117. Example.....	225
118. Example.....	226
119. Example.....	226
120. Example.....	226
121. Example.....	226
122. Example.....	226
123. Example.....	227

124. Example.....	227
125. Example.....	227
126. Example.....	227
127. Example.....	227
128. Example.....	228
129. Example.....	228
130. Example.....	228
131. Example.....	228
132. Example.....	229
133. Example.....	229
134. Example.....	229
135. Example.....	229
136. Example.....	229
137. Example.....	230
138. Example.....	230
139. Example.....	230
140. Example.....	230
141. Example.....	230
142. Example.....	231
143. Example.....	231
144. Example of JCL to Use the APPCLOG Utility.....	239

---

# Tables

1. Example of Multi-trans Resource Processing.....	31
2. Parameters Used in TP Message Log Definition.....	34
3. Combination of Parameters Used in TP Message Log Definition.....	35
4. Summary of TP Profile Keywords.....	64
5. Summary of Side Information Keywords.....	67
6. APPC/MVS Administration Utility Commands and Required Data.....	75
7. TP Profile Administration.....	83
8. Side Information Administration.....	84
9. Database Token Administration.....	84
10. Variables in ICQASE00.....	90
11. How APPC sets byte 5 of the Exchange Log Name (X'1211') GDS Variable.....	113
12. Persistent Verification Verb Exit Return and Reason Codes.....	158
13. General Format of SIGNON/Change Password GDS Variable.....	162
14. Format of GDS Variable Sub-fields For A Sign-on Request.....	162
15. Format of GDS Variable Sub-fields For A Sign-on and Change Password Request.....	162
16. Format of GDS Variable Sub-fields For SIGNON/Change Password Reply.....	163
17. Example GDS Variable (SIGNON/Change password TP input).....	163
18. GDS Variable Structure (SIGNON/Change password TP output).....	164
19. SIGNON/Change Password TP Status Values.....	164
20. SIGNON/Change Password Request Formatting Errors.....	165
21. Security Mechanisms Available, based on Application Security Types.....	180
22. Summary of RACF Classes and Resource Names.....	181
23. VTAM's HALT Command and Its Effect on APPC/MVS Work.....	193

24. Stopping APPC/MVS Work.....	194
25. Displaying TP Status.....	196
26. Displaying UR Status.....	197
27. Displaying Server Status.....	197
28. Displaying LU Status.....	198
29. Displaying Scheduling Status.....	198
30. SMF Records for APPC/MVS.....	211
31. Local LU and TP profile level interactions.....	216
32. Character Sets 01134, Type A, and 00640.....	233



## About this information

---

This information supports z/OS (5650–ZOS).

APPC/MVS is an implementation of IBM's Advanced Program-to-Program Communication (APPC) in the MVS operating system. APPC/MVS allows MVS application programs to communicate on a peer-to-peer basis with other application programs on the same z/OS system, different z/OS systems, or different operating systems including Microsoft Windows®, Sun Solaris, AIX®, OS/400®, OS/2, and VM in an SNA network. These communicating programs, known as **transaction programs** or TPs, together form cooperative processing applications that can exploit the strengths of different computer architectures. Whenever the term **program** is used in this information, it means transaction program, unless otherwise stated.

In this information, the term **transaction program** refers to a program scheduled by the APPC/MVS transaction scheduler (ASCH) or to any other program, running in an MVS address space, that uses APPC/MVS services. The term **transaction** is not restricted to programs scheduled by the APPC/MVS transaction scheduler, or to programs using APPC/MVS services.

Note that APPC/MVS transaction programs are parts of cooperative processing applications and are distinct from, but coexistent and compatible with, CICS® and IMS transaction processing applications.

This publication contains the general information necessary to plan, configure, activate, customize, and maintain APPC/MVS. When more detailed or related information exists elsewhere, this publication references other publications.

See [\*z/OS Planning for Installation\*](#) for information about how to install the software products that are necessary to run APPC/MVS.

## Who should use this information

---

This information is for the following people:

- System programmers responsible for setting up an APPC network that includes APPC/MVS.
- APPC administrators who define transaction programs to APPC/MVS
- Security administrators responsible for the security of APPC transaction programs.

## How to use this information

---

All users of this information should read [Chapter 1, “Introduction to APPC/MVS,” on page 3](#) to familiarize themselves with APPC as it is implemented in MVS. [Chapter 2, “Planning Overview,” on page 19](#) introduces program management, session management, security management and operations for APPC/MVS. This publication describes each of these management areas as follows:

- [Part 2, “Program management,” on page 25](#)
  - [Chapter 3, “Scheduling Transaction Programs,” on page 27](#) describes the special scheduling considerations for APPC/MVS transaction programs.
  - [Chapter 4, “Defining Scheduling Characteristics with ASCHPMxx,” on page 43](#) describes how to use the ASCHPMxx parmlib member to define scheduling characteristics for transaction programs that use the APPC/MVS transaction scheduler.
  - [Chapter 5, “Controlling the Execution of Transaction Programs,” on page 55](#) describes how to define transaction programs to MVS through TP profiles and side information.
  - [Chapter 6, “Using the APPC/MVS Administration Utility,” on page 73](#) describes how to use the APPC/MVS administration utility to define TP profiles and side information.
  - [Chapter 7, “Using the APPC/MVS Administration Dialog,” on page 83](#) describes the panel interface used in the APPC/MVS administration dialog, which defines TP profiles and side information.

- [Part 3, “Session management,” on page 93](#)
  - [Chapter 8, “Planning Sessions,” on page 95](#) describes defining sessions using VTAM® in an SNA network.
  - [Chapter 9, “Controlling Configuration through APPCPMxx,” on page 121](#) tells how to use the APPCPMxx parmlib member to define APPC/MVS local LUs and their session characteristics.
- [Part 4, “Security management,” on page 131](#)
  - [Chapter 10, “Setting up Network Security,” on page 133](#) describes how to use RACF® to set up APPC network security.
- [Part 5, “System management,” on page 183](#)
  - [Chapter 11, “Operating APPC/MVS,” on page 185](#) describes how to use MVS operator commands to initialize and maintain APPC/MVS.
  - [Chapter 12, “APPC/MVS Measurement and Tuning,” on page 209](#) describes how to set up performance groups for APPC, measure how APPC work performs in the system, and then tune various aspects of the system to optimize how APPC uses system resources.
- [Part 6, “Installation checklists,” on page 223](#)
  - [Chapter 13, “Installing an APPC Application,” on page 225](#) describes the basic installation requirements for installing APPC, including the requirements for installing a TP that initiates an outbound request and for installing a TP that responds to an inbound request.

The last section contains installation information and includes an install checklist for two types of transaction programs. Items in the checklist refer back to previous sections in the publication for details.

## Where to find more information

Where necessary, this publication references information in other publications, using the shortened version of the publication title. For complete titles and order numbers of the publications for all products that are part of z/OS, see [z/OS Information Roadmap](#). The following table lists the titles and order numbers of publications for other IBM products.

Short Title Used in This publication	Title	Order Number
<i>AS/400 APPC Programmer's Guide</i>	<i>AS/400 Communications: Advanced Program-to-Program Communication Programmer's Guide</i>	SC41-8189
<i>CPI-C Reference</i>	<i>Common Programming Interface Communications Reference</i>	SC26-4399
<i>OS/400 Communications Configuration Reference</i>	<i>AS/400 Communications: Operating System/400® Communications Configuration Reference</i>	SC41-0001
<i>SNA Formats</i>	<i>SNA Formats</i>	GA27-3136
<i>SNA LU 6.2 Reference: Peer Protocols</i>	<i>SNA Network Architecture LU 6.2 Reference: Peer Protocols</i>	SC31-6808
<i>SNA Network Product Formats</i>	<i>SNA Network Product Formats</i>	LY43-0081
<i>SNA Technical Overview</i>	<i>SNA Technical Overview</i>	GC30-3073
<i>SNA Transaction Programmer's Reference Document for LU 6.2</i>	<i>SNA Transaction Programmer's Reference Document for LU 6.2</i>	SC31-6808
<b>VM</b>		
<i>VM/ESA Connectivity</i>	<i>VM/ESA Connectivity</i>	SC24-5756

## How to provide feedback to IBM

---

We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information. For more information, see [How to send feedback to IBM](#).



## Summary of changes

---

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

**Note:** IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy) ([www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy](http://www.ibm.com/docs/en/zos/latest?topic=zos-product-documentation-update-policy)).

## Summary of changes for z/OS 3.2

---

The following content is new, changed, or no longer included in z/OS 3.2.

### New

The following content is new.

#### September 2025 release

- None.

### Changed

The following content is changed.

#### September 2025 release

- None.

### Deleted

The following content is deleted.

#### September 2025 release

- None.

## Summary of changes for z/OS 3.1

---

The following content is new, changed, or no longer included in z/OS 3.1.

### New

The following content is new.

#### September 2023 release

- None.

### Changed

The following content is changed.

#### May 2025 refresh

- A restriction is added. See [“Starting the APPC and ASCH Address Spaces”](#) on page 185.

#### September 2023 release

- None.

## **Deleted**

The following content was deleted.

### **September 2023 release**

- None.

---

## Part 1. Introduction





# Chapter 1. Introduction to APPC/MVS

## References:

- *SNA Network Concepts and Products*
- *CPI-C Reference*

## APPC Overview

Advanced Program-to-Program Communication (APPC) is an implementation of the Systems Network Architecture (SNA) LU 6.2 protocol on a given system. APPC allows interconnected systems to communicate and share the processing of programs.

## How APPC Relates to SNA, LU 6.2, VTAM, and CPI-C

Many organizations require fast and accurate exchanges of data to perform their business functions, and they depend on communication networks to facilitate such data exchange. To address data processing and communication needs, IBM designed the SNA architecture as a guide for connecting products in a communications network.

The SNA architecture provides formats and protocols that define a variety of physical and logical SNA components. One such logical component, called the logical unit (LU), is responsible for handling communication between end users and provides each end user with access to the network. SNA defines different types of logical units to meet the needs of specific end users, whether the end user is an application program, a stand-alone terminal, or a terminal and an operator. *LU 6.2* is a type of logical unit that is specifically designed to handle communications between application programs.

Figure 1 on page 3 depicts a logical view of an SNA network that handles communication from different users through LUs.

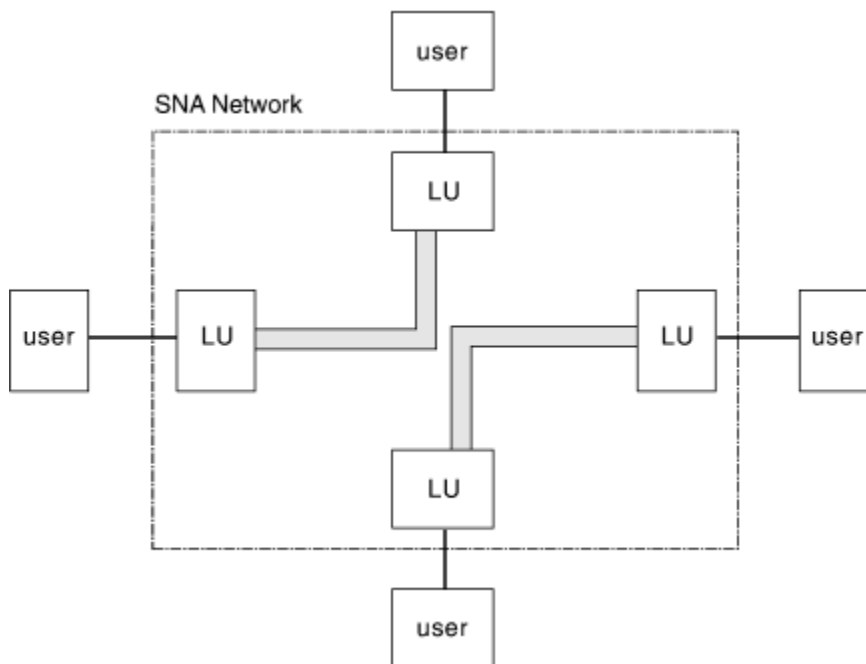
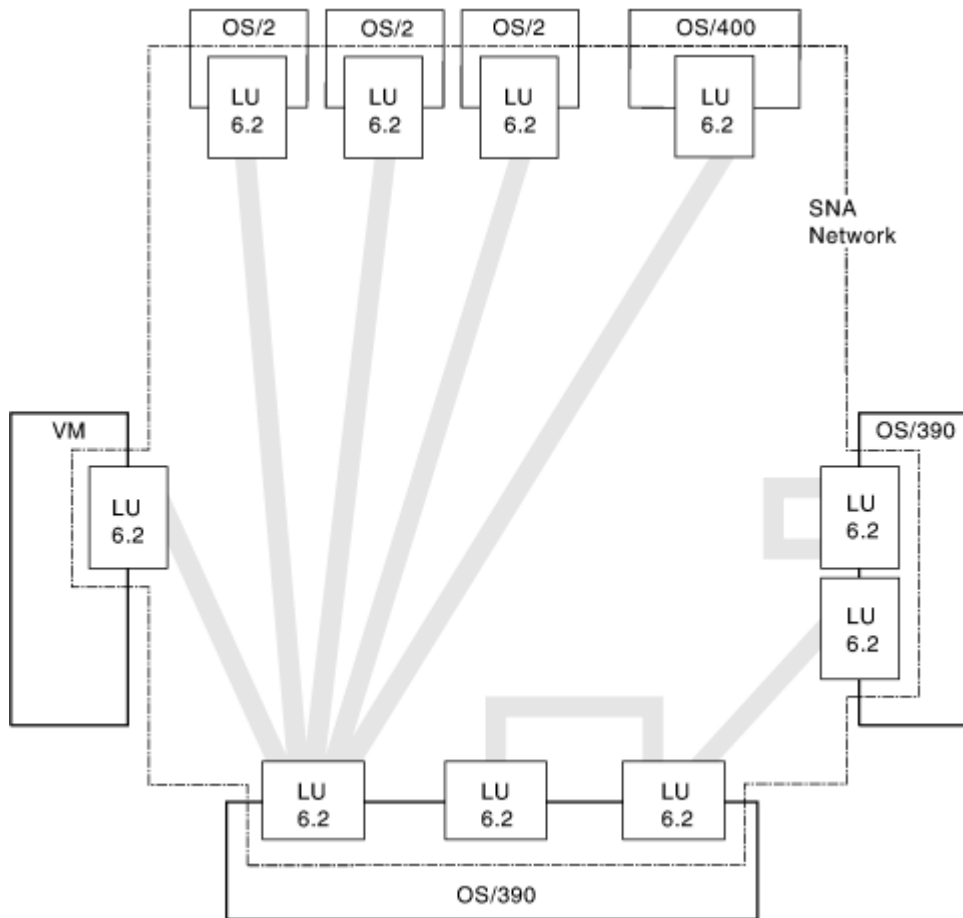


Figure 1. Network Communications between LUs and Users

A typical SNA network consists of a diverse collection of processors or nodes. Some nodes may be running the z/OS or VM operating systems. Using LU 6.2, an APPC application running on one of these

processors can communicate with a remote APPC application running on another processor, regardless of the type of processor on which the remote application is running.

A product that makes such communication possible between applications on diverse processors is *Virtual Telecommunications Access Method (VTAM)*. VTAM and APPC/VTAM are implementations of SNA architecture, which direct data between programs and devices. [Figure 2 on page 4](#) represents an SNA network that is directing data among unlike systems.



*Figure 2. An SNA Network for Communications between Different Systems*

Before data can flow over a network, the application programs that cause the exchange of data must request communication services. For programmers who code these applications, it is desirable to use a consistent interface to the communications services, regardless of the environment. To address the need for a consistent interface across different environments, IBM introduced Common Programming Interface Communications (CPI-C). CPI-C defines how applications written in high-level languages can be integrated and ported across various platforms, such as z/OS, OS/390, AS/400, VM/ESA, and workstations.

The following example ([Figure 3 on page 5](#)) represents a network application of two transaction programs (A and B) that use CPI Communications calls to establish the APPC type of communication called a conversation. The conversation is directed by the CMxxxx calls, which initialize and allocate the conversation (CMINIT and CMALLC), send and receive data (CMSEND and CMRCV), and eventually deallocate the conversation (CMDEAL).

The sample conversation shown could represent an application in which a workstation program (Program A) sends input to its partner in z/OS (Program B), which then processes and stores the input in a database.

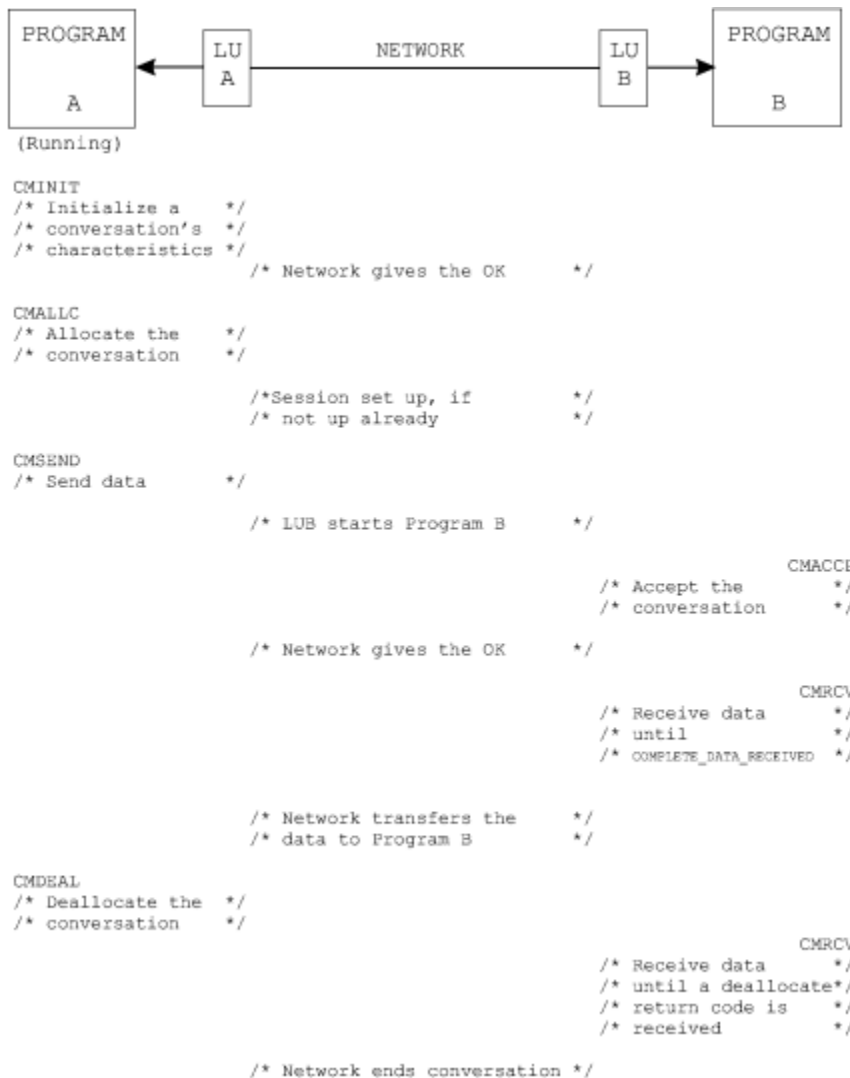


Figure 3. CPI Communications Program Scenario

## APPC Concepts and Commonly Used Terms

Before planning an APPC network, you must be familiar with certain SNA terms as used in APPC. The APPC/MVS implementation of LU 6.2 uses the common SNA programming and network terms that follow.

### Programming Terms

#### Transaction Program (TP)

An application program that uses APPC communication calls is a transaction program, or TP. A TP on one system can communicate with a TP on another system to access resources on both systems. Both TPs can be considered a single cooperative processing application that happens to reside on two different systems.

#### Local TP/Partner TP

Whether a TP is a local TP or a partner TP usually depends on point of view. From the point of view of a z/OS system, TPs residing on the system are local TPs, and TPs on remote systems are partner TPs. However, from the point of view of the remote system, the names are reversed: the TPs that reside on its system are local TPs and the ones on z/OS are the partner TPs.

A local TP can initiate communication with one or more partner TPs. The partner might or might not reside on the local system. The TP does not need to know whether the partner TP is on the same system or on a remote system.

Other terms for TPs are *inbound TP* and *outbound TP*, which convey who establishes the communication. An outbound TP is the one that starts a conversation and an inbound TP is the one that responds. In [Figure 3 on page 5](#), program A is the outbound TP and program B is the inbound TP. On z/OS, any program that calls APPC/MVS services to start a conversation is considered an outbound TP, while an inbound TP requires special processing by z/OS, such as scheduling and initiation, or processing by an APPC/MVS server.

### Client TP

A client transaction program is one that requests the services of an APPC/MVS server.

### APPC/MVS Server

An APPC/MVS server is an MVS application program that uses the APPC/MVS Receive\_Allocate callable service to receive allocate requests from one or more client TPs. An APPC/MVS server can serve multiple requestors serially or concurrently.

### Conversation

The communication between TPs is called a conversation. Like a telephone conversation, one TP calls the other and they “converse,” one TP “talking” at a time, until one TP ends the conversation. The conversation uses predefined communication services that are based on SNA-architected LU 6.2 services called verbs. These verb services are implemented in APPC/MVS as callable services.

To start (allocate) a conversation, a TP issues an allocate call that contains specific information, such as the name of the partner TP, the LU in the network where the partner TP resides, and other network and security information. The conversation is established when the partner TP accepts the conversation. After a conversation is established, other calls can transfer and receive data until a TP ends (deallocates) the conversation with a Deallocate call.

**Note:** The CPI Communications protocol requires an Initialize\_Conversation (CMINIT) call before an Allocate call.

### Conversation\_ID

A conversation\_ID is an 8-byte token that the Allocate, Initialize\_Conversation, Accept\_Conversation, and Receive\_Allocate calls return. APPC provides the conversation\_ID to uniquely identify the conversation on subsequent APPC calls.

### TP\_ID

A TP\_ID is a unique 8-byte token that APPC/MVS assigns to each instance of an inbound transaction program. When multiple instances of a TP are running simultaneously under APPC/MVS, they have the same TP name, but each has a unique TP\_ID. The TP\_ID can be used to trace a specific instance of a TP in the system.

### Conversation State

To ensure orderly conversations and prevent both TPs from trying to send or receive data at the same time, APPC enforces conversation states. TPs enter specific conversation states by calling specific APPC services, and the states determine what services the TP may call next. For example, when a local TP allocates a conversation, the local TP is initially in send state; and when the partner TP accepts the conversation, the partner is in receive state. As the need arises, the local TP can call a receive service to enter receive state and put its partner in send state, allowing the partner to send data.

### Inbound/Outbound Allocate Request

An inbound allocate request is one that starts a conversation with a TP on z/OS; an outbound allocate request is a request to start a conversation from a local TP on z/OS.

### Inbound/Outbound Conversation

Whether a conversation is inbound or outbound, similar to whether a TP is a local TP or a partner TP, depends on point of view. From the point of view of an z/OS system, an inbound conversation originates from a TP that issues an inbound allocate request for a TP on the z/OS system. An outbound conversation originates from a TP on the z/OS system that issues an outbound allocate request for its partner.

The significant difference between inbound and outbound conversations generally has to do with whether the conversation will initiate work that requires special processing by z/OS. Inbound

conversations might allocate local TPs on z/OS that need to be scheduled by a transaction scheduler, or inbound conversations might need to be queued for an APPC/MVS server.

## Network Terms

### Logical Units (LUs) and LU 6.2

A logical unit is an SNA addressable unit that manages the exchange of data and acts as an intermediary between an end user and the network. There are different types of logical units. Some LU types support communication between application programs and different kinds of workstations. Other LU types support communication between two programs. LU type 6.2 specifically supports program-to-program communication. The actual implementation of LU 6.2 on a given system is APPC.

### Local LU/Partner LU

Whether an LU is a local LU or a partner LU depends on point of view. From the point of view of an z/OS system, LUs defined to the z/OS system are local LUs and LUs defined to remote systems are partner LUs. However, from the point of view of the remote system, the names are reversed: the LUs that are defined to its system are local LUs and the ones on z/OS are the partner LUs.

A partner LU might or might not be on the same system as the local LU. When both LUs are on the same system, the LU through which communication is initiated is the local LU, and the LU through which communication is received is the partner LU.

LUs are defined to VTAM on z/OS by APPL statements in SYS1.VTAMLST. LUs managed by APPC/MVS must also be defined by LUADD statements in APPCPMxx parmlib members.

### Physical units (PUs)

An SNA network consists of physical nodes, called physical units (PUs), which are connected by physical data links. Typically in an APPC/MVS network, there is at least one host node (PU 5), one communication controller node (PU 4), and any number of peripheral nodes (PU 2.0 and PU 2.1). A PU 2.0 can be either a programmable workstation or a non-programmable workstation that is configured as a dependent LU. A dependent LU can support a single session. Because only one conversation can flow over a session, the dependent LU can be used by only one transaction program at a time. A PU 2.1 is a physical node, such as a programmable workstation, an AS/400, or other compatible hardware, that can be configured as an independent LU. An independent LU can support multiple sessions concurrently. With a conversation flowing over each session, the LU can be used by many transaction programs at the same time.

### Sessions

A session is a logical connection that is established or *bound* between two LUs of the same type. A session acts as a conduit through which data moves between the pair of LUs.

The following figure shows how a session spans two LUs that are defined on two different systems.



Figure 4. A Session between Two LUs

A session can support only one conversation at a time, but one session can support many conversations in sequence. Because sessions are reused by multiple conversations, a session is a long-lived connection compared to a conversation.

If no session exists when a TP issues an Allocate call to start a conversation, VTAM binds a session between the local LU and the partner LU. After a session is bound, TPs can communicate with each other over the session in a conversation. This sending of data between a local TP and its partner occurs until one TP ends the conversation with a Deallocate call.

The following figure shows a single conversation between TP1 and TP2 that is occurring over a session.

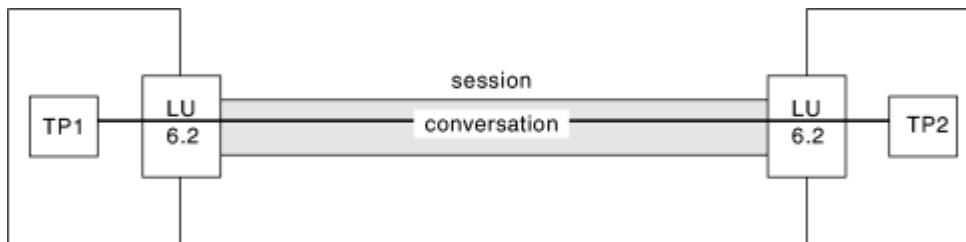


Figure 5. A Conversation between Two TPs

If the hardware permits and the two LUs are configured as independent LUs, they can have multiple, concurrent sessions called *parallel sessions*. When a TP from either LU issues an allocate call and sessions exist but are being used by other conversations, an LU can request a new session unless the defined session limit is reached.

Default session limits are defined for an LU in a VTAM APPL statement. Session-limit values can be changed by entering the VTAM MODIFY CNOS and MODIFY DEFINE operator commands, or by modifying the VTAM APPL definition statement and then restarting APPC/MVS. For more information about these commands, see *z/OS Communications Server: SNA Operation*.

The following figure shows three parallel sessions, each of which is carrying a conversation.

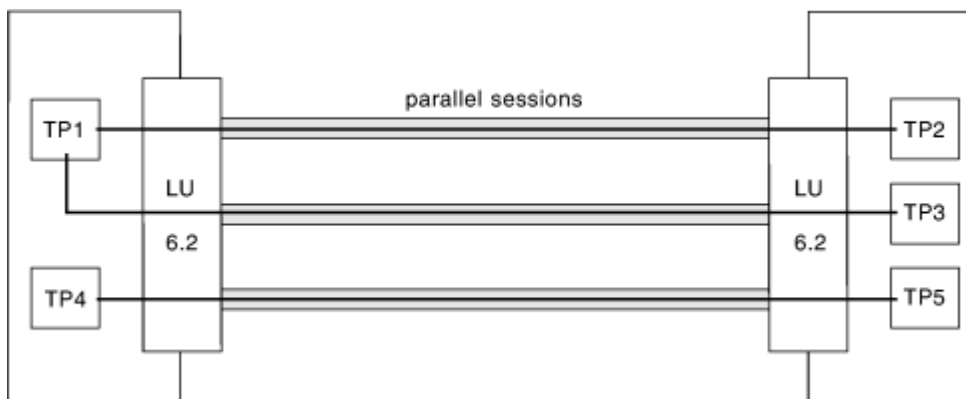


Figure 6. Parallel Sessions between LUs

An installation can define different types of sessions, but sessions are ultimately defined by the LUs they span and by the session characteristics contained in the VTAM logon mode table that is associated with the session.

Sessions can span LUs on the same system, LUs on two like systems, and LUs on two unlike systems that are LU 6.2 compatible. The following figure shows three sessions bound from a single LU on SYS2. Session 1 spans LUs on two different systems. Session 2 spans the same two systems but is bound from a different LU on SYS1. Session 3 is bound between two LUs on the same system.

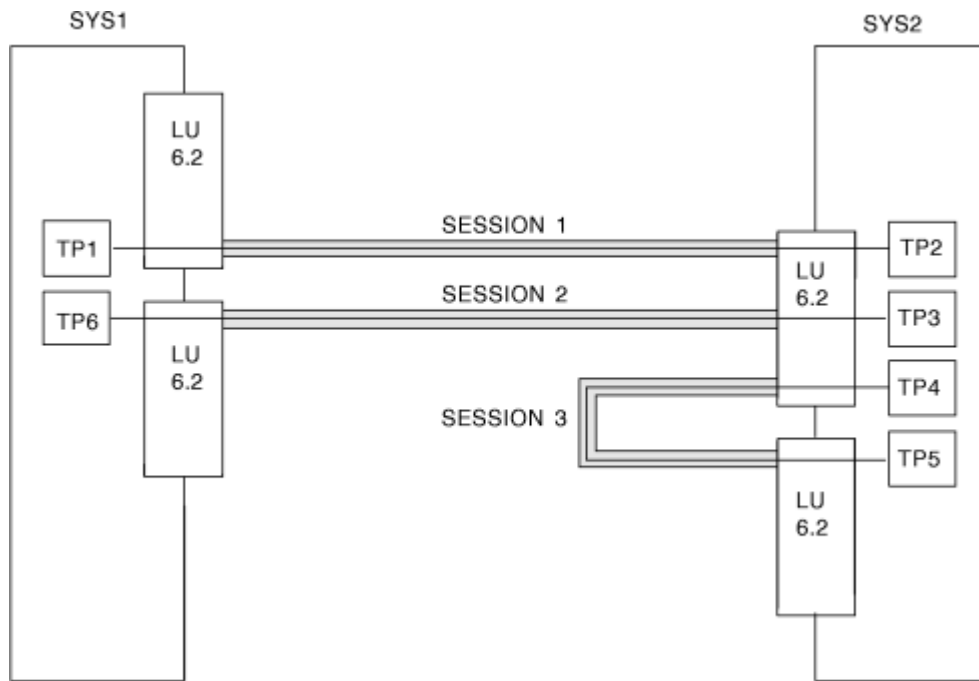


Figure 7. Different Types of Sessions between Two LUs

### Logon Modes

A logon mode contains the parameters and protocols that determine a session's characteristics. Logon modes are defined in a VTAM logon mode table, a compiled version of which exists in SYS1.VTAMLIB. Session characteristics, such as pacing levels and class of service, are controlled by entries in logon mode tables. A VTAM system programmer can use these characteristics to define sessions for specific types of transactions, such as for ASCII data, satellite communication, high-speed transactions, batch, and interactive data.

### Contention

When a TP from each LU in a session simultaneously attempts to start a conversation, the situation that results is called contention. To control which TP can allocate the conversation, a system programmer can define for each LU the number of sessions in which it is the *contention winner* and the number of sessions in which the LU is the *contention loser*.

Generally, a system programmer divides the contention winner role between the two LUs. For example, if two LUs can have ten parallel sessions and if the sessions will be equally started by both LUs, each LU is designated the contention winner for five sessions and the contention loser for five sessions. A contention winner can use the session without informing the contention loser. A contention loser can request use of the session from the contention winner.

Default contention winners and losers are defined for an LU in a VTAM APPL statement.

## What is APPC/MVS?

APPC/MVS is a VTAM application that extends APPC support to the z/OS operating system. Although APPC/VTAM previously provided some LU 6.2 capability, APPC/MVS in cooperation with APPC/VTAM provides full LU 6.2 capability to programs running in z/OS.

The primary role of APPC/MVS is to provide a set of MVS callable services that enable z/OS application programs to communicate with other application programs through communication protocols provided by the SNA network.

APPC/MVS consists of programming support and z/OS system support. The programming support consists of APPC/MVS callable services and administrative system files for transaction programs. The z/OS system support enables programs to use the callable services in z/OS.

## Programming Support for APPC/MVS Callable Services

The APPC/MVS callable services can be divided into five types as shown in [Figure 8 on page 10](#). Each type is explained in more detail following the figure.

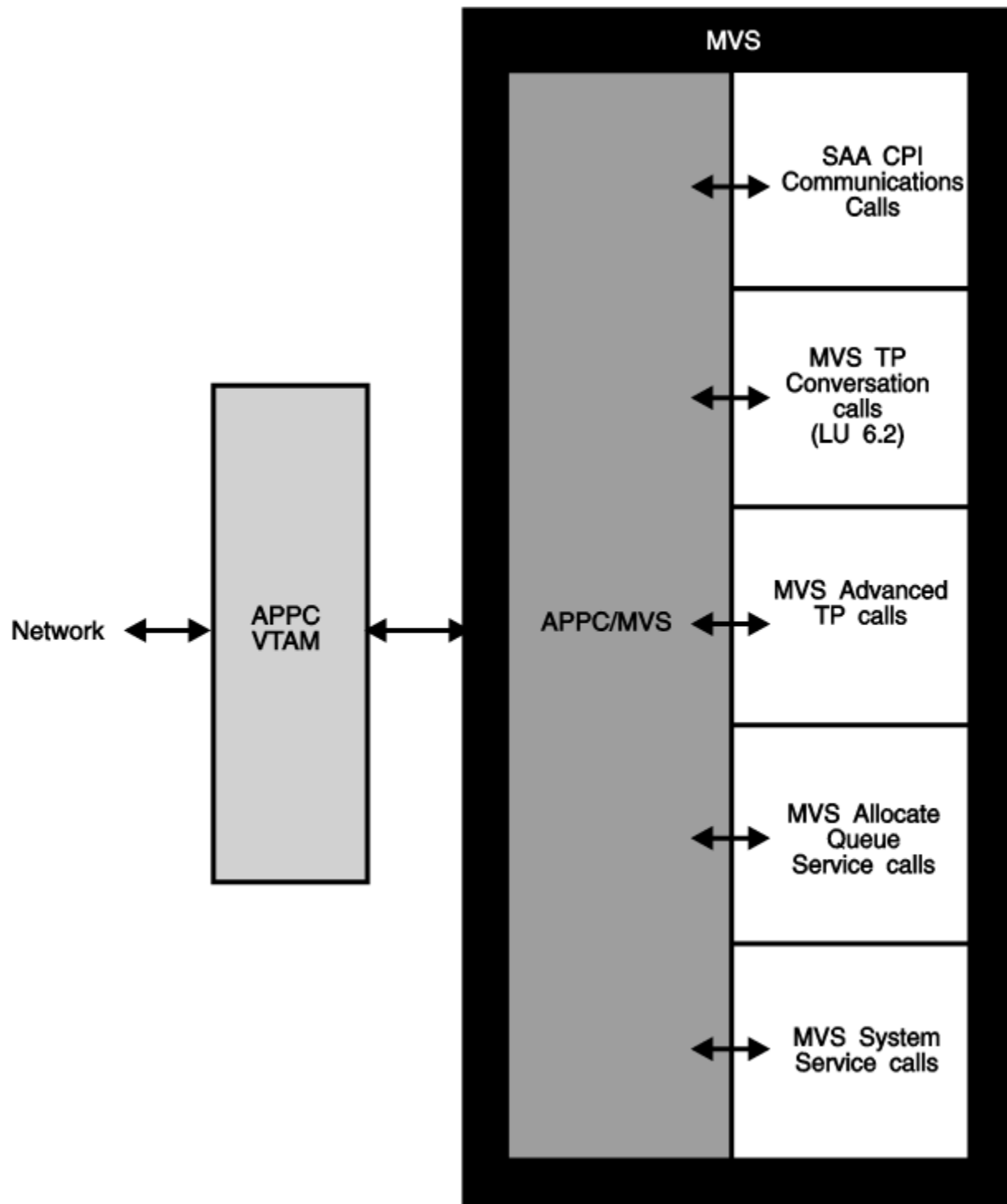


Figure 8. Types of APPC/MVS Callable Services

### CPI Communications Calls

Common Programming Interface (CPI) Communications calls allow high-level language programs to communicate regardless of the system on which they are running. High-level language programs use the CPI Communications calls to establish conversations and pass data back and forth. When



programs in z/OS use these calls, the underlying implementation may be different from another system, but the results are equivalent.

For example, a distributed application written in C could have part of the application on a workstation configured for APPC and the other part on an z/OS system running APPC/MVS. The two parts of the application could communicate using the same CPI Communications calls, even though their underlying environments are different. Programs that use only the CPI Communications calls can be ported to many other systems.

The CPI Communications calls use the SNA LU 6.2 architected verbs. Each communication call is prefixed by the letters CM; for example, CMALLC (Allocate). For more information, including languages supported, see *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*.

### **APPC/MVS TP Conversation Calls**

The APPC/MVS TP conversation calls are the z/OS implementation of the SNA LU 6.2 architected verbs and are prefixed by the letters ATB. These conversation calls are similar to the CPI Communications calls except that the z/OS versions take advantage of specific z/OS functions. For example, the z/OS Send\_Data call (ATBSEND) can send data residing in a data space—something the CPI Communications Send call cannot do.

Like the CPI Communications calls, the APPC/MVS TP conversation calls can be issued from a high-level language such as COBOL, C, PL/I, FORTRAN, and REXX, or from assembler language programs.

Unlike the CPI Communications calls, programs issuing the z/OS calls are not portable to other systems.

### **APPC/MVS TP Advanced Calls**

The z/OS advanced TP calls provide unique, non-LU 6.2 architected services to TPs running in z/OS. These calls provide specific z/OS functions, such as the ability to extract information about communications resources used by APPC/MVS transaction programs.

The advanced calls can be issued from high-level languages other than REXX, and from assembler language programs.

### **APPC/MVS Allocate Queue Services Calls**

The APPC/MVS allocate queue services calls allow a server address space on z/OS to own and manage inbound allocate requests. Servers own allocate requests by *registering* for them through the Register\_For\_Allocates callable service.

Rather than directing such requests to a transaction scheduler, APPC/MVS places allocate requests for which a server has registered on a structure called an allocate queue. APPC/MVS queues allocate requests on a first-in, first-out (FIFO) basis. Servers process allocate requests by selecting them from allocate queues and performing the requested function.

The allocate queue services, which can be called from a high-level language such as COBOL, C, PL/I, FORTRAN, and REXX, or from assembler language programs, are described in *z/OS MVS Programming: Writing Servers for APPC/MVS*.

The allocate queue services calls are not based on the LU 6.2 architecture.

### **APPC/MVS System Service Calls**

Another type of APPC/MVS callable service provides access to system services not normally used by transaction programs. These services are used by other z/OS components, subsystems, and transaction schedulers, which run in supervisor state or PSW key 0-7. The system services calls can be called from assembler and high-level languages other than REXX, and are documented in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

The z/OS system service calls are not based on the LU 6.2 architecture.

## **Administrative System Files**

In addition to the callable services, APPC/MVS programming support provides an administrative utility that creates and maintains entries about TPs (TP profiles and side information) in Virtual Storage Access Method (VSAM) key sequenced data sets (KSDS). The entries in the VSAM system files provide

information that facilitates the flow of conversations across sessions. The two types of entries are placed in different VSAM files—a TP profile file and side information file.

A TP profile file contains scheduling and security information for z/OS programs that are scheduled in response to inbound allocate requests. Each LU is assigned a TP profile file that contains information about the programs that will be associated with that LU. When an LU receives an inbound allocate request, it locates in its TP profile file the information necessary to retrieve and schedule the transaction program requested. A TP profile file can be assigned to more than one LU at a time.

Inbound allocate requests for which a server has registered are not scheduled, and therefore do not require a TP profile.

The side information file contains the translation of symbolic destination names used by:

- z/OS local TPs, when issuing outbound allocate requests
- APPC/MVS servers, when registering for inbound allocate requests.

If the allocate or register request does not specify a symbolic destination name, other parameters with routing information must be specified. There can be only one side information file per system in use at one time.

### ***Use of TP Profile and Side Information for a Scheduled Conversation***

Figure 9 on page 12 shows how TP profile and side information files are used by TPs on two different systems. TP1 on the peer system allocates a conversation across the network to TP2, using symbolic destination name TP2sym. The side information translates TP2sym into the necessary information to send the allocate request to the correct LU on z/OS and to the correct TP profile. The TP profile schedules TP2 to run so it can accept the allocate request with a Get\_Conversation call. TP2 then allocates a different conversation across the network to TP3 using symbolic destination name TP3sym, and the process repeats itself going from z/OS to the peer system.

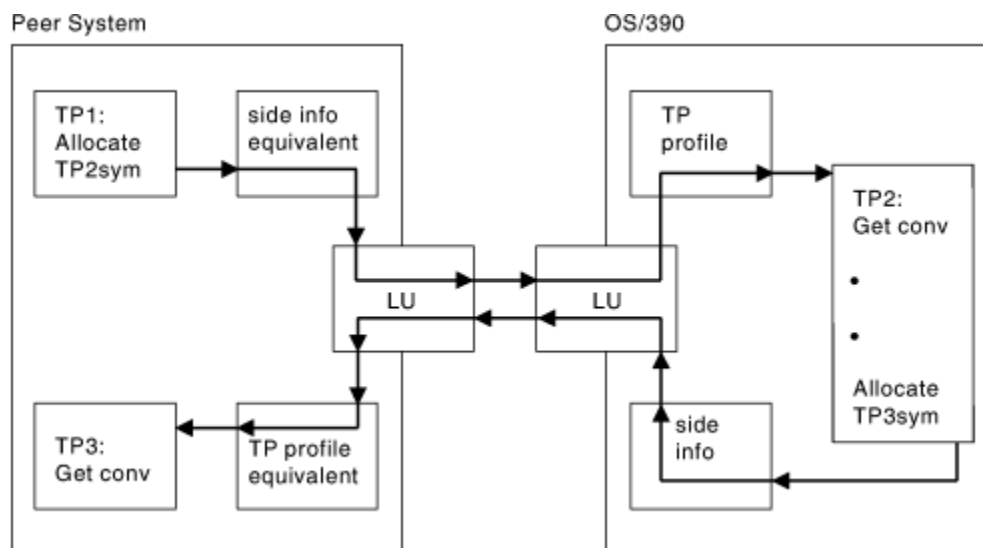


Figure 9. Using TP Profiles and Side Information to Find a Partner TP

### ***Use of Side Information for an APPC/MVS Server***

Figure 10 on page 13 shows how side information files are used by a client TP and its APPC/MVS server on two different systems. The client TP on the peer system allocates a conversation across the network to the server, using symbolic destination name SERVsym. The side information on the peer system translates SERVsym into the necessary information to send the allocate request to the server. Note that served requests do not require the use of a TP profile.

Before APPC/MVS can queue the allocate request for the server, the server must have previously registered for the request through the Register\_For\_Allocates service. When it registered, the server specified symbolic destination name TPsym on the call to Register\_For\_Allocates to own inbound

conversations from the client TP. The side information on the z/OS system translated TPsym into the necessary information to identify allocate requests from the client TP. The server receives the conversation through the Receive\_Allocate service so that APPC communications can ensue between client and server.

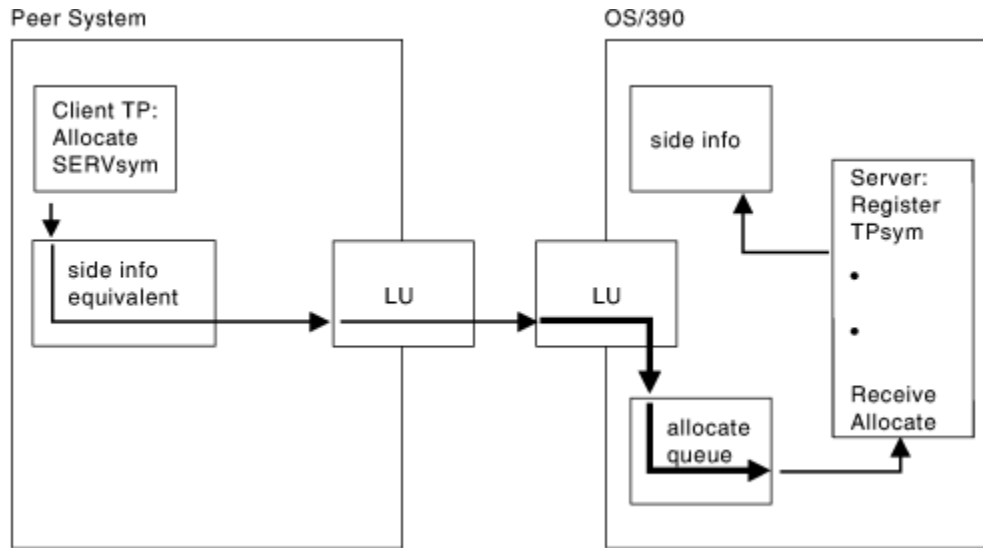


Figure 10. Using Side Information in Client/Server Communications

A system programmer uses APPC administration utility (ATBSDFMU) to maintain the TP profile and side information files by submitting a batch job that can add, modify, retrieve, and delete entries. An interactive panel dialog using the APPC administration utility is available with TSO/E 2.3 and above.

For more information about the administrative utility and the dialog, see Chapter 5, “Controlling the Execution of Transaction Programs,” on page 55 *z/OS MVS Planning: APPC/MVS Management*.

## z/OS System Support

APPC/MVS operates primarily in two startable MVS address spaces, APPC and ASCH. The APPC/MVS communication functions run in the APPC address space and the APPC/MVS transaction scheduler functions run in the ASCH address space.

Transactions residing in z/OS can be scheduled by the APPC/MVS transaction scheduler or by an installation-defined scheduler. Transactions can also be routed directly to an APPC/MVS server address space, rather than being scheduled.

When the APPC/MVS transaction scheduler is used, the installation can:

- Assign TPs to classes with specific scheduling characteristics.
- Assign TPs to a schedule type of *standard* or *multi-trans*. Standard scheduling allocates resources for each transaction and deallocates them when the TP ends. Multi-trans scheduling causes a transaction program to remain active between inbound conversations with its resources available. This type of scheduling avoids the overhead of repeated resource allocation and deallocation.

If an installation or product requires a specialized scheduler, APPC/MVS provides system services that allow you to write a customized transaction scheduler, or to specify a scheduler in addition to the APPC/MVS transaction scheduler. However, before using an alternate transaction scheduler, you should first investigate using an APPC/MVS server.

## Specific APPC/MVS Support

Specific system support for APPC/MVS consists of:

- Two parmlib members, one to contain definitions for the APPC address space (APPCPMxx) and the other to contain definitions for the APPC/MVS transaction scheduler address space (ASCHPMxx).

### **APPCPMxx**

Names APPC/MVS local logical units (LUs) and the administrative VSAM KSDSs. It also sets up a correspondence between local LUs and transaction schedulers, and, optionally, defines LUs that are not to be associated with schedulers ("NOSCHED" LUs).

### **ASCHPMxx**

Defines classes for the APPC/MVS transaction scheduler that determine scheduling characteristics, such as the response time goal for TPs running in a class.

- APPC implementation through MVS system commands:
  - START command initiates the APPC and ASCH address spaces.
  - SET command specifies the parmlib member or members whose definitions are to dynamically modify APPC and ASCH characteristics.
  - DISPLAY command displays current communication configurations, information about queued and running transactions, information about active conversations, and information about APPC/MVS server address spaces and allocate queues.
  - CANCEL command terminates local TPs and cancels the APPC and ASCH address spaces.
  - STOP command terminates an initiator address space.
  - TRACE, TRACK, and STOPTR help with recovery and diagnostics associated with APPC/MVS TPs.
- Interactive problem control system (IPCS) subcommands, APPCDATA and ASCHDATA, that aid in problem determination for APPC/MVS.
- System resource manager (SRM) performance management of TPs scheduled under the APPC/MVS transaction scheduler and based on installation performance criteria for all TPs, individual TPs, TP initiator classes, or account numbers.
- Resource Measurement Facility (RMF) monitoring of TP performance.
- System management facility (SMF) accounting for conversations by any TP and for resources used by TPs scheduled under the APPC/MVS transaction scheduler.

## **Overview of an APPC/MVS Outbound Request**

When a local TP makes a request to establish a conversation with its partner, the request is called an "outbound" request.

Figure 11 on page 15 illustrates APPC/MVS initialized and ready to service communication requests. Communications services are available through application programming interfaces to any MVS address space, such as TSO/E users, batch jobs, and started tasks. An application (local TP) running in any existing MVS address space can allocate a conversation with a partner TP. Note that the APPC/MVS transaction scheduler plays no role in outbound requests.

If a symbolic destination name was used to allocate a conversation, the side information file is accessed to translate the symbolic destination name into the required routing information.

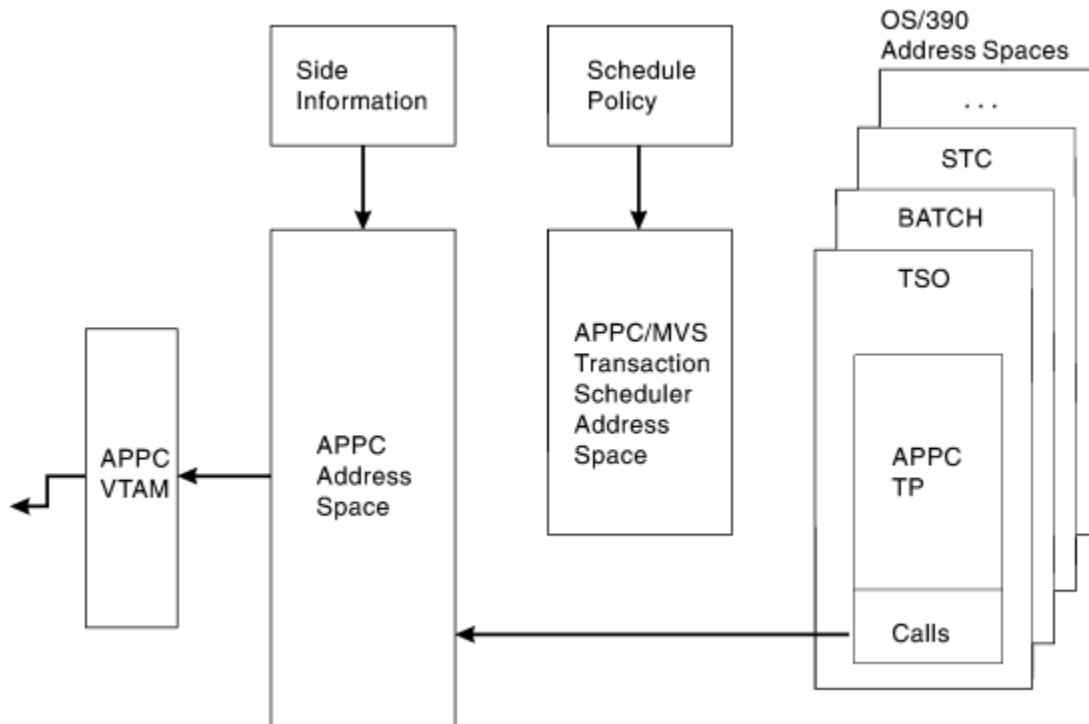


Figure 11. APPC/MVS Communications Services (Outbound)

## Overview of an APPC/MVS Inbound Request

When a request to establish communications comes from a remote node in the network into the local z/OS system, it is called an “inbound” request. An inbound request could also come from the same LU.

An illustration of inbound processing follows.

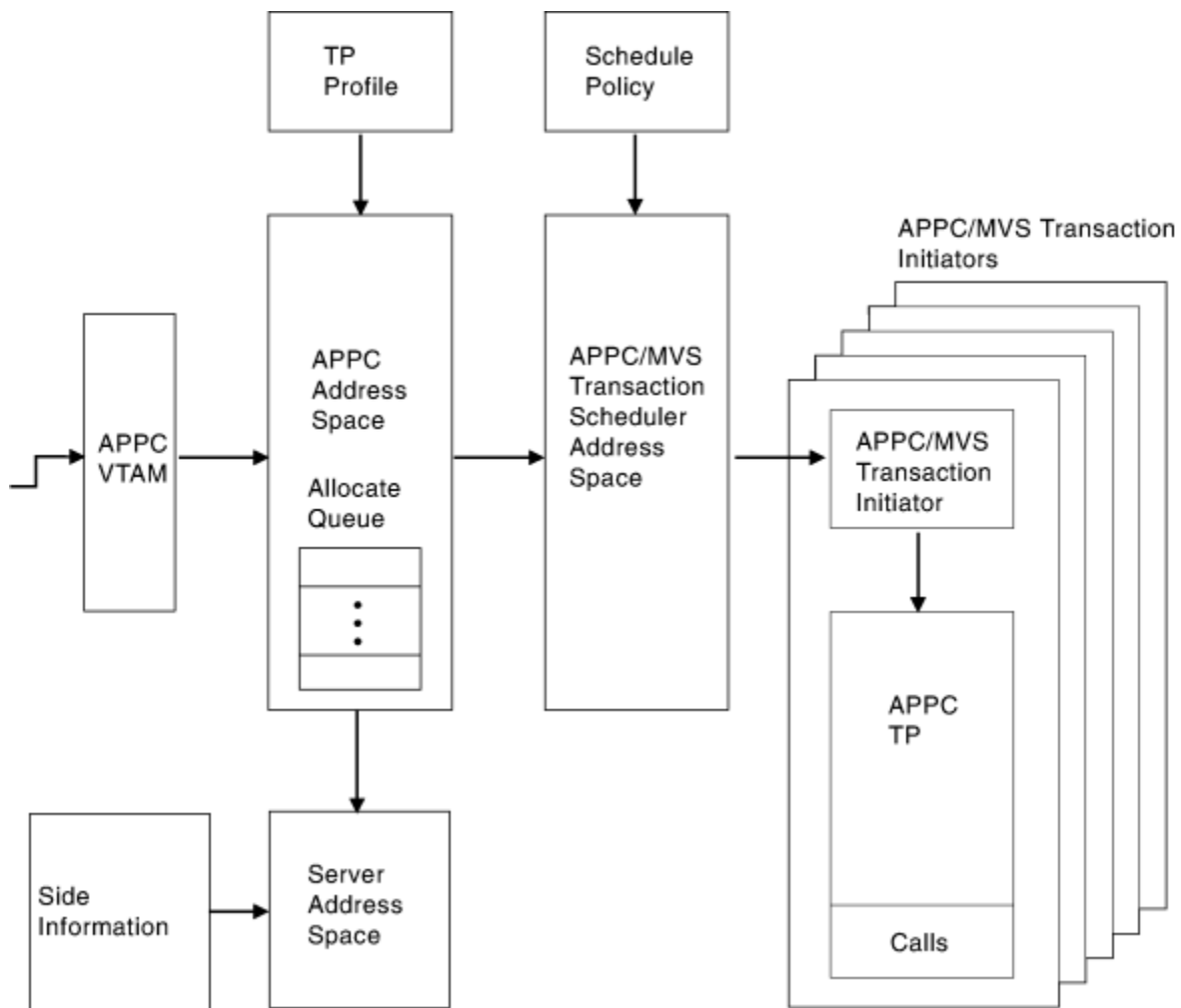


Figure 12. APPC/MVS Communication Services (Inbound)

An installation can use Resource Access Control Facility (RACF) or an equivalent security product to check that the inbound request is authorized to access the local LU. A security environment can then be established to validate access to other resources.

The inbound request contains the 1- to 64-character name of the local TP that is to be attached. When an inbound request enters the system, APPC/MVS first checks to see whether any address spaces on the local system had previously requested to serve the request (that is, whether an APPC/MVS server has registered for the request through the Register\_For\_Allocates service). If so, APPC/MVS places the request on an allocate queue from which the server can later select it for processing. When the server selects the request from the allocate queue, it receives the conversation ID, and a conversation with the issuer of the request starts.

If the server used a symbolic destination name to register for the request, APPC/MVS uses the side information file to translate the symbolic destination name into the required routing information.

If no servers have registered for the request, APPC/MVS attempts to schedule the request to a transaction scheduler. APPC/MVS maps the name of the TP targeted by the request to a TP profile that contains information necessary to set up the appropriate z/OS environment that will be required to run the TP. All inbound TPs processed by the APPC/MVS transaction scheduler *must* have a TP profile associated with them. The TP profile contains information such as:

- Transaction program capabilities and status
- Transaction scheduler information:
  - MVS job name

- MVS program name (for example, “IEBMAIL”)
- Data set allocation environment
- Execution class.

The APPC/MVS transaction scheduler is responsible for maintaining pools of address spaces into which TPs are scheduled. These address spaces can receive the services of all MVS components, and are called subordinate address spaces. An APPC *transaction initiator* is the program that runs in each of the APPC/MVS transaction scheduler's subordinate address spaces, and is responsible for setting up the appropriate environment (as specified in the TP profile) and managing the processing of the TPs. The APPC/MVS transaction initiator is similar to the MVS initiator that provides a processing environment for traditional types of work on z/OS (such as batch jobs). The term *transaction initiator* is used throughout this document to mean an APPC/MVS transaction scheduler subordinate address space. [Figure 12 on page 16](#) shows these initiators on the right-hand side.





# Chapter 2. Planning Overview

An APPC network is transaction-driven. Thus, the physical and logical aspects of the network reflect the types of transaction programs that will communicate over the network. For example, if an installation wants to run an office application with the user interface on several PCs and the storage and search facilities on an z/OS system, the necessary levels of connection between the PCs and the z/OS system must be set up. Once the network structure is in place, program communication can flow much like conversations on a telephone line.

Planning and defining an APPC network is a team effort involving MVS system programmers as well as application programmers, VTAM programmers, security administrators, hardware engineers, and system programmers for other systems in the network. Planning usually begins with a clear idea of the kinds of programs that will run in the network. Then, to allow the programs to communicate, appropriate participants define levels of connections in MVS and VTAM, and make corresponding connection definitions on the peer systems.

Reference:

*[z/OS Communications Server: SNA Network Implementation Guide](#)*

## Levels of Connections

Before programs on two different systems can converse in an APPC network, connections on various levels must exist. Basically the connections are on the physical level, the logical level, and the program level. The physical level of connection is the hardware that links like and unlike systems together. After the physical connection is made, a logical level of connection is possible. Logical connections are the definitions that permit VTAM to establish a session. The session allows transaction programs on different systems to communicate on the program level. The program level connection is the transaction program identification and the use of callable services, which establish a conversation.

A hardware connection can support multiple sessions. A single session supports one conversation at a time. [Figure 13 on page 19](#) shows the relationship of the three levels of connection.

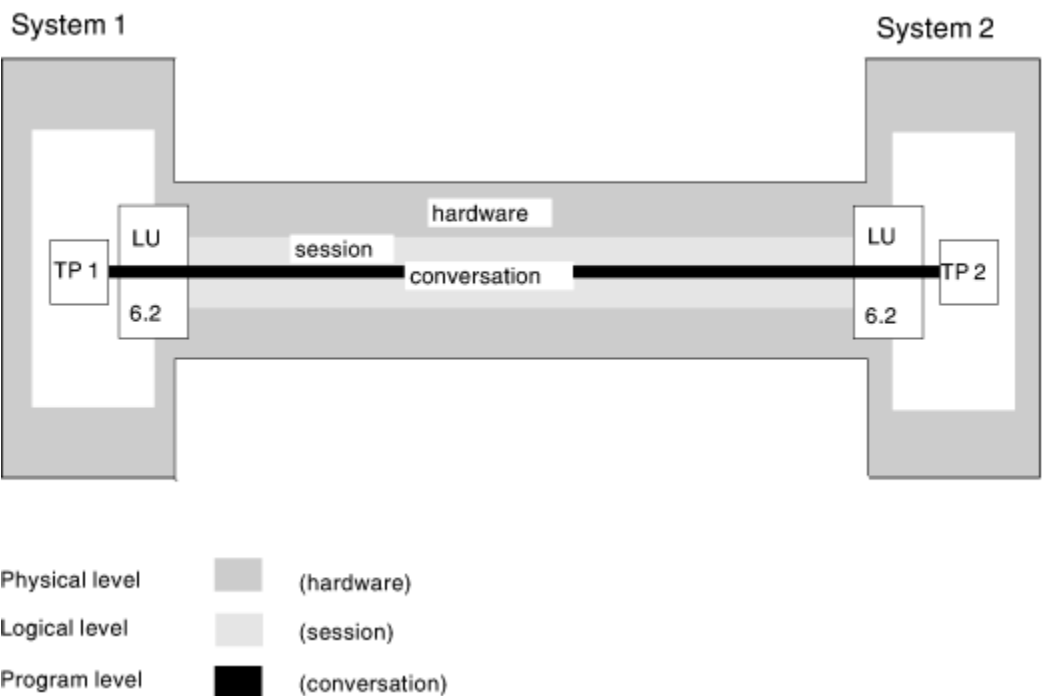


Figure 13. Levels of Connections

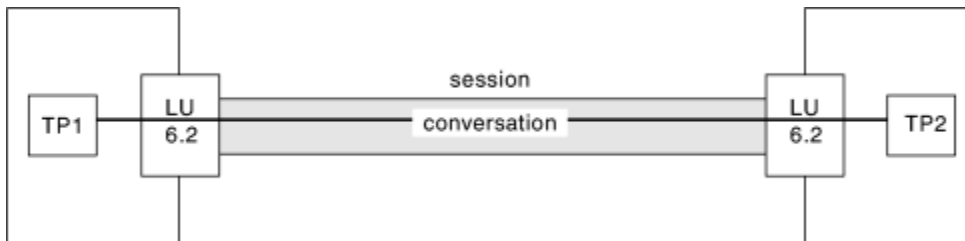
When using RACF, a security administrator can define security for an APPC network on both the logical level and the program level. Physical level security is accomplished through facilities planning.

For information about security, see [Chapter 10, “Setting up Network Security,”](#) on page 133.

## Physical Connections

There are many hardware configurations that support APPC/MVS. For example, several combinations of hardware can link a workstation to an z/OS system. The configuration you choose depends on your intended use of APPC/MVS, the hardware available, and the physical connections already in place.

[Figure 14 on page 20](#) shows a sample configuration to give you an idea of some connectivity options available through APPC/MVS.



*Figure 14. APPC Connectivity Options*

The configuration example shows a connection of two z/OS systems and the connection of one of the z/OS systems to many workstations through a communications controller.

Hosts can be connected through a channel-to-channel adapter. The two hosts would be running MVS/ESA SP 4.2 or higher with an appropriate level of VTAM. In the example, one z/OS system has two LUs defined: SRV1LU01 and SRV1LU02. The other z/OS system has one LU defined, SRV2LU03. Each z/OS system belongs to its own SNA network subarea.

To connect the host to the workstations, a communications controller from the 37x5 family is used. The 37x5 communications controller is installed between the host and the workstation to allow remote communications through Synchronous Data Link Control (SDLC). The 37x5 must be running an appropriate level of NCP. If the 37x5 has a Token Ring interface card (TIC) installed, a Token Ring of workstations can be connected to the 37x5 directly or through a gateway workstation.

An alternative configuration for remote communications is to connect a workstation to a modem, and communicate over the phone lines to another modem, which would then be connected to the 37x5. The 37x5 and all of the connected workstations make up another subarea. Each workstation has its own LU.

Another possible configuration is to arrange multiple workstations in a Token Ring network with a gateway workstation connected to the 37x5 through SDLC and two modems.

Transaction programs communicating through any of the defined LUs can communicate with each other. A transaction program on SRV1LU01 can communicate with a transaction program on SRV1LU02 or SRV2LU03. TPs can communicate between the two z/OS systems, or between an z/OS system and any of the workstations.

## Program Connections

Transaction programs that run in an APPC/MVS network must follow general APPC requirements as well as MVS requirements. For example, one general requirement is that any TP starting a conversation in an APPC network must identify its logical connection by directly or indirectly naming a logon mode and the partner LU.

These and other program level considerations are described in [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

## Logical Connections and APPC/MVS Management

Before a TP running on an z/OS system can use APPC/MVS to converse, system programmers must provide the information z/OS needs to support the TP's conversations. Communication requirements must be determined and translated into LUs and VTAM session definitions. In addition, TPs must be defined to the system and grouped according to scheduling classes. If the conversations require security, RACF provides various methods to secure all or part of the supporting APPC/MVS structure.

These considerations and many more fall under the category of APPC/MVS management, the topic of this document.

## APPC Management Tasks

---

The management tasks required to support APPC/MVS fall into four categories:

### Program Management

After an installation has coded or acquired APPC applications, a programmer defines scheduling classes for the TPs and uses the administration utility or dialog to define TP profiles and side information.

### Session Management

To enable TPs to communicate over the network, an installation must plan each LU 6.2 it needs and the characteristics of sessions that connect the LU to LU 6.2s on other systems. Each LU 6.2 is defined to APPC/MVS as a local LU, and to VTAM in an APPL statement. Session definitions are determined by logon mode names that must first be defined in VTAM and must correspond to session characteristics defined on the partner system.

### Security Management

Security of APPC/MVS can be controlled on the LU-to-LU level, LU-to-TP level, TP-to-TP level, LU-to-user level, and TP-to-user level. Some specific ways of controlling access are through defining TPs as RACF resources, which controls access to TPs, and defining a pair of LUs as a RACF resource, which allows specific levels of conversation security.

### System Management

To operate APPC/MVS, use various MVS system commands such as START, DISPLAY, SET, STOP, and CANCEL. To control how APPC work performs in relation to other z/OS work, create an SRM performance group for TPs scheduled by the APPC/MVS transaction scheduler and a second SRM performance group for any transactions processed by APPC/MVS servers. You can also use SMF records to audit TPs.

The remainder of this document is organized into four parts based on these management categories. The way the tasks are ordered is not, of course, the only way to order them; and the processes described are not performed only when APPC is first introduced on an z/OS system. The tasks and processes are ongoing. As an installation acquires more TPs, for example, those TPs must be defined, possibly creating the need to add an LU or change an existing LU.

## System-Wide APPC Connections

Figure 15 on page 23 shows the MVS support structure for APPC/MVS using specific values. The examples, which show only a subset of all possible situations, go across the four management areas using values that are numbered. A description of each numbered value follows:

- **1** Name of APPC/MVS transaction scheduler class

A class of transaction initiators for the APPC/MVS transaction scheduler must be defined in an ASCHPMxx member of the parmlib concatenation. A TP is assigned to a class through a TP profile. An APPC/MVS transaction scheduler class is recorded in an SMF type 33 subtype 1 record.

- **2** Name of VSAM KSDS for TP profiles

TP profiles are contained in a VSAM key sequenced data set (KSDS). You must first define the VSAM KSDS with a DEFINE CLUSTER command and then associate it to an LU in an LUADD statement within

an APPCPMxx member of the parmlib concatenation. If you need RACF protection for the VSAM KSDS, define it with a RACF ADDSD command and then allow access to specifically named users.

- **3a** Transaction program name

A transaction program name can be from 1 to 64 characters long. It is a required parameter in the TP profile key and may be, but does not have to be, the same as the jobname in the TP profile JCL. The transaction program name is recorded in an SMF type 33 record (subtypes 1 and 2). If you need RACF protection for an individual TP, you can define the transaction program name to a RACF APPCTP class. If you specify TPNAME in the side information, it must match the TPNAME in the TP profile, or the TP will not run. So, in [Figure 15 on page 23](#), the entry for TPNAME in SYS1.APPCTP must match the entry for TPNAME in SYS1.APPCSI.

- **3b** Jobname

A job name in the TP profile is 1 to 8 characters long and is the name of the JCL job that runs the transaction program. The job name can be, but does not have to be, the same as the transaction program name used in the TP profile key. To cancel a transaction program, use the job name and address space identifier in a CANCEL command.

- **4** Level of access for a transaction program

Transaction program execution access is determined by the level of the TP profile and the level of the LU through which the transaction program communicates. RACF can use the level of access to specifically define access to each transaction program and its TP profile.

- **5a 5b** Account number

An account number can be assigned to a transaction program in TP profile JCL (5a) or in a RACF user profile (5b). When the TP's account is to be tailored, the account number comes from the RACF user profile. SMF can audit a TP using either account number.

- **6** Database token for VSAM KSDS

You can assign a database token to each VSAM KSDS to represent the file name in RACF security definitions that give access to TPs.

- **7** Name of VSAM KSDS for side information

Side information is contained in a VSAM key sequenced data set (KSDS). You must define the VSAM KSDS with a DEFINE CLUSTER command and then associate it with an LU in a SIDEINFO statement within an APPCPMxx parmlib member. If you need RACF protection for the VSAM KSDS, define it with a RACF ADDSD command and then allow access to specifically named users.

- **8** Logon mode name

A logon mode contains the definition of session characteristics for network communication. Logon modes are compiled entries in the VTAM logon mode table in SYS1.VTAMLIB. (The logon mode in this example is shown as uncompiled source.) When a transaction program initiates a conversation, it must specify a logon mode or indirectly specify it. In APPC/MVS, a TP can indirectly specify a logon mode through a reference to side information.

- **9** Partner LU name

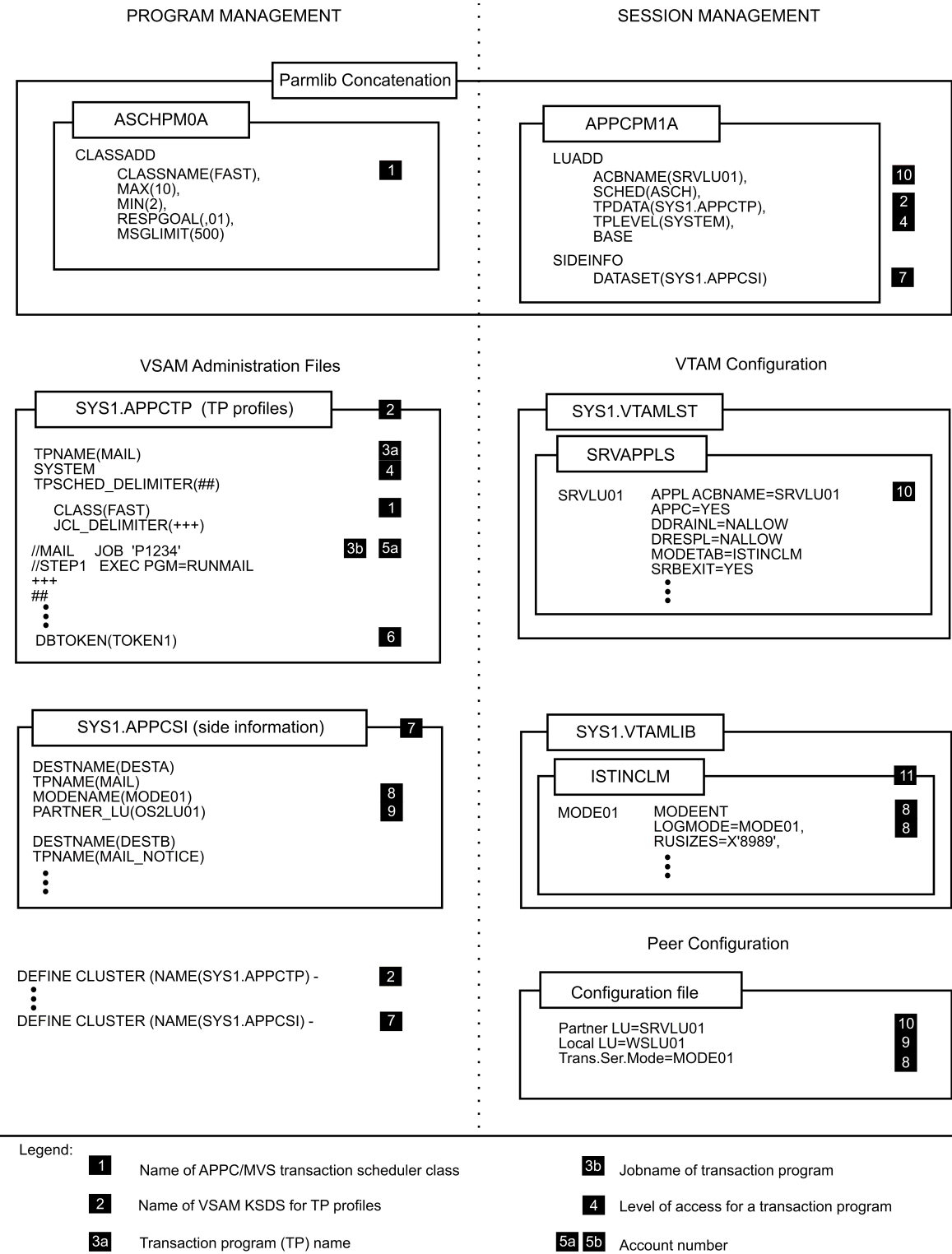
The partner LU (logical unit) is the SNA type 6.2 LU that, in this example, is located on a workstation. When an outbound transaction program initiates a conversation, it can directly or indirectly specify the partner LU. In z/OS, a TP can indirectly specify the partner LU in side information. SMF records the partner LU name in SMF type 33 records (subtypes 1 and 2). If RACF protection is required, the partner LU is specified in RACF LU access definitions.

- **10** Local LU name

The local LU (logical unit) is the SNA type 6.2 LU that, in this example, is located on an z/OS system. In APPC/MVS, the local LU is defined by an LUADD statement in an APPCPMxx parmlib member. In VTAM, the z/OS local LU is defined by a VTAM APPL definition statement in SYS1.VTAMLST.

- **11** Name of VTAM logon mode table

The VTAM logon mode table contains logon mode entries that define network session characteristics. The logon modes available for an z/OS local LU are contained in the table named after the MODETAB parameter in the LU's VTAM APPL statement.



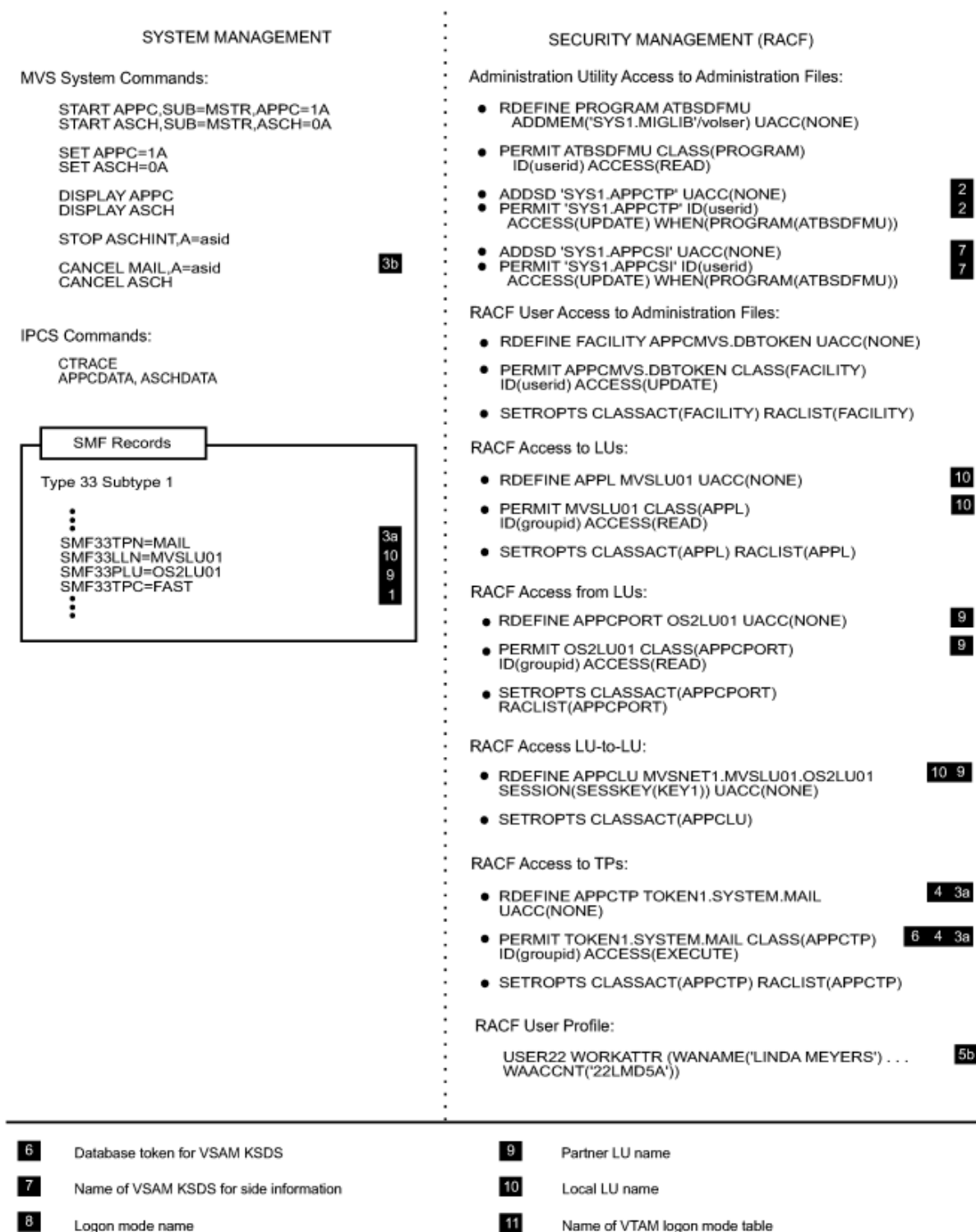


Figure 16. System-Wide APPC Connections (Part 2 of 2)

---

## Part 2. Program management





---

## Chapter 3. Scheduling Transaction Programs

MVS processes inbound APPC/MVS transaction programs (TPs) differently from other types of work. To initiate and schedule TPs in response to inbound requests from their partners, APPC/MVS provides a transaction scheduler separate from the job entry subsystems.

References:

*z/OS MVS System Management Facilities (SMF)*

---

### Overview of Transaction Scheduling

In terms of transaction scheduling, there are two distinct types of APPC transaction programs in MVS, namely outbound and inbound. A transaction program that initiates a conversation by issuing an allocate request is an outbound transaction program; a transaction program that receives the allocate request is an inbound transaction program.

Inbound transaction programs are different from batch jobs and started tasks, because the transaction programs are introduced into the system by a work scheduler that is different from the output processor. In contrast, traditional work is controlled from scheduling to output processing by a job entry subsystem.

APPC/MVS provides a transaction scheduler that initiates and schedules APPC/MVS TPs in response to inbound allocate requests for conversations. APPC/MVS also provides system services that let an installation run TPs under a different transaction scheduler. Those system services are applicable to MVS subsystems or other application environments that provide their own work schedulers.

A transaction scheduler commonly has direct control over a number of address spaces and schedules its applications into these "subordinate" address spaces; the use of subordinate address spaces allows a transaction scheduler to access APPC from its own environment for additional performance and function. Each transaction scheduler may have its own term for these subordinate address spaces; for example, the APPC/MVS transaction scheduler refers to them as "transaction initiators."

See [Figure 17 on page 28](#) for an overview of how multiple transaction schedulers and their subordinate address spaces operate under APPC/MVS. When APPC/MVS receives an inbound allocate request for a particular LU, it sends a message describing the request to the associated transaction scheduler. That scheduler then initiates the appropriate transaction program.

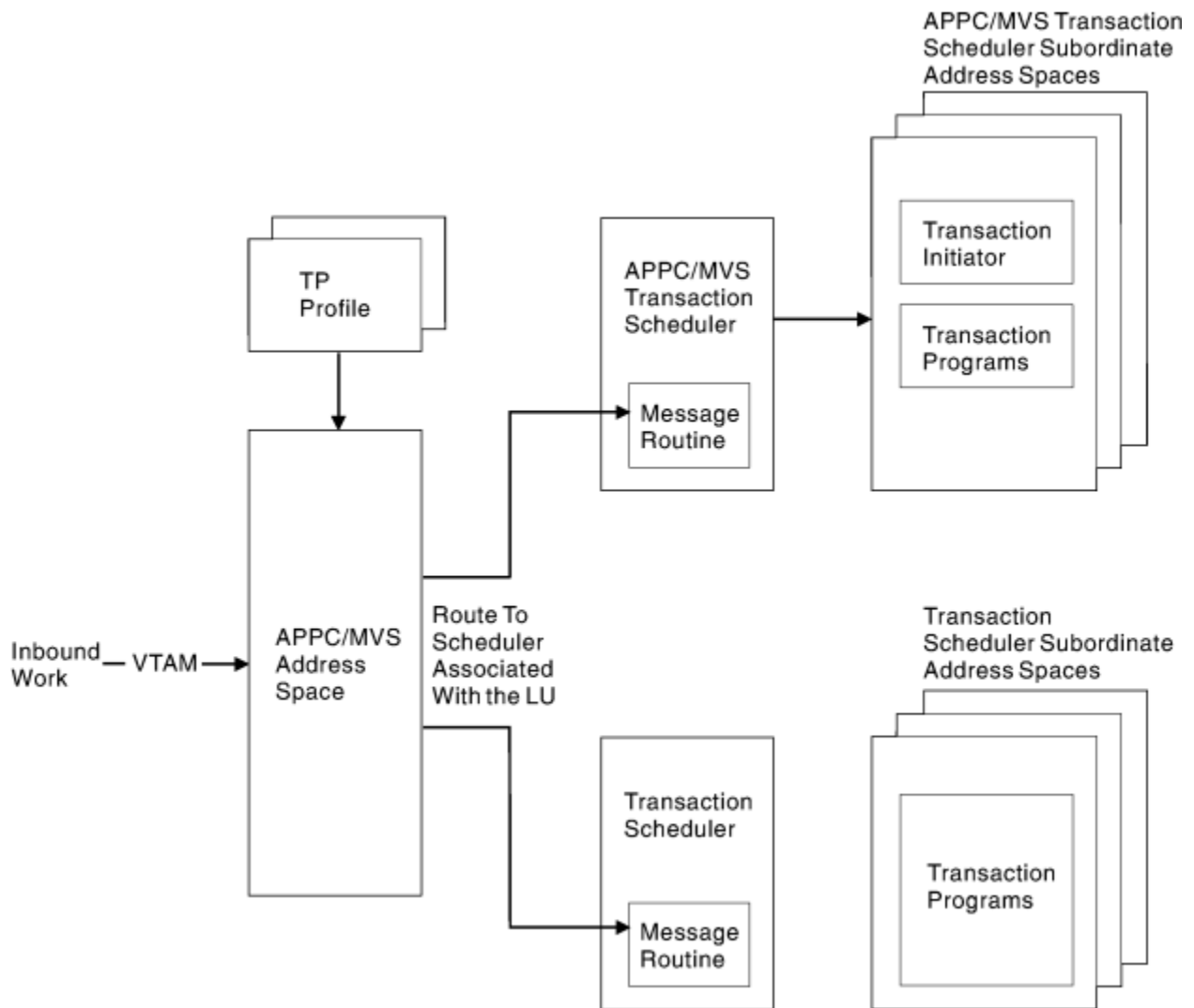


Figure 17. Transaction Program Routing

## Scheduling Characteristics of the APPC/MVS Transaction Scheduler

When an inbound APPC/MVS transaction program is scheduled by the APPC/MVS transaction scheduler, it is distinguished from other MVS work as follows:

### Initialization

In response to an inbound allocate request from its partner, a transaction program is initiated through a program controlled interface, namely the allocate callable service. If the allocate service follows the LU 6.2 protocol, it can be issued from a program on the local MVS and from a program in a different operating system or even in a different network. For information about the Allocate call and other callable services used in APPC, see [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

### Definition

Each inbound APPC/MVS transaction program is defined by a TP profile that is created by the installation and stored in a VSAM key sequenced data set. The TP profile contains scheduling parameters and a subset of JCL to define resource requirements. For information about TP profiles, see Chapter 5, “Controlling the Execution of Transaction Programs,” on page 55. For information about the subset of JCL to use, see [“Specific Scheduler JCL Information for TP Profiles”](#) on page 67

### Output

APPC/MVS-scheduled TPs can use the same JES SYSOUT functions available to other MVS applications, with the exception that SYSOUT data sets allocated by TPs are treated as spin data

sets. To guarantee that spin data sets are processed before initiator cleanup, IBM recommends that all SYSOUT specifications, for both standard and multi-trans TPs, include the FREE=CLOSE parameter. For more information about SYSOUT processing, see [“Specific Scheduler JCL Information for TP Profiles”](#) on page 67.

If the installation changes the default subsystem for APPC work with the SUBSYS parameter in an ASCHPMxx parmlib member, then JES services may be unavailable to APPC/MVS-scheduled TPs.

### Performance

Transaction initiators are managed based on response time goals set for classes. Classes are defined in an ASCHPMxx parmlib member, and a TP is assigned to a class in its TP profile. More information about classes appears later in this chapter and in [Chapter 4, “Defining Scheduling Characteristics with ASCHPMxx,”](#) on page 43. Information about how to set a response time goal appears in [“Defining Classes and Response Time Goals”](#) on page 214.

For information about APPC performance, see [Chapter 12, “APPC/MVS Measurement and Tuning,”](#) on page 209.

### Management

System management facilities (SMF) provides records to support transaction programs running on an z/OS system, and the Resource Measurement Facility (RMF) reports on this work. For more information about the support SMF provides, see [“Using SMF to Audit APPC Work”](#) on page 210.

### Security

Authority to use a transaction program can be controlled by a security product. Transaction programs can be made available as a public resource, or can be limited to particular users or groups of users. For more information, see [Chapter 10, “Setting up Network Security,”](#) on page 133.

### Problem Determination

Transaction program processing is logged differently from work scheduled by a job entry subsystem. Runtime execution messages are written to a TP message log, the name of which is specified in the TP profile. For more information about logging, see [“Logging Transaction Program Processing”](#) on page 33.

When APPC/MVS finds errors during processing of inbound requests, APPC/MVS can send error log information to a partner system or program. Error log information describes errors that APPC/MVS finds when it tries to schedule a TP, such as problems with logical unit (LU) definitions or problems with processing allocate requests. Alternate transaction schedulers can also send error log information when processing their inbound requests.

For information about how an alternate scheduler can send error log information and the format of the variable that is used to send that information, see the description of the Cleanup\_TP service in [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#).

## Using the APPC/MVS Transaction Scheduler

---

The APPC/MVS transaction scheduler is a startable address space that is initialized by a START ASCH command at the time the APPC address space is initialized. To use the APPC/MVS transaction scheduler, both the APPC and ASCH address spaces must be started. (For information about starting these address spaces, see [“Starting the APPC and ASCH Address Spaces”](#) on page 185.)

When a TP runs under the APPC/MVS transaction scheduler, it can take advantage of:

- Classes of transaction initiators
- DISPLAY ASCH operator command to monitor scheduling activity
- TP schedule types

## Classes of Transaction Initiators

The APPC/MVS transaction scheduler can schedule programs according to classes that an installation defines in a parmlib member (ASCHPMxx) and assigns to programs in TP profiles. For each class, the

installation can assign a minimum and maximum number of transaction initiators to be available for initiating programs and assign a response time goal for each TP in the class.

Classes should be defined based on the characteristics of the TPs that will run in a particular class. TPs for a particular class should have similar characteristics such as run-time, priority, schedule-type, and security. The APPC/MVS transaction scheduler attempts to optimize scheduling within the limits set by the system programmer, by varying the number of transaction initiators according to the work load level and reassigning transaction initiators to other classes as the need arises. For more information about defining classes, see [Chapter 4, “Defining Scheduling Characteristics with ASCHPMxx,”](#) on page 43.

**Note:** At least one class definition is required for work assigned to the APPC/MVS transaction scheduler.

## DISPLAY Command

The DISPLAY ASCH command can monitor the activity of the APPC/MVS transaction scheduler. The information displayed and the degree of detail are controlled through various parameters. It is possible to display the current number of classes, number of active TPs, queued TPs, and the total number of transaction initiators as well as the ones that are idle. For more information about the DISPLAY command, see [“Displaying Information about APPC/MVS Work”](#) on page 186.

## TP Schedule Types

Transaction programs initiated by the APPC/MVS transaction scheduler are divided into two scheduling types: standard and multi-trans. The schedule type controls how easily the program can be re-invoked.

### Standard Schedule Type

When transaction programs are scheduled as standard (the default), APPC/MVS initializes them for each inbound conversation request and terminates them when they finish processing. With standard scheduling, a transaction program's resources are allocated and deallocated for each inbound conversation request. Standard scheduling provides a clean environment each time the TP is scheduled, and isolates TPs from each other and from subsequent requests for the same transaction program. The standard schedule type provides full security, data integrity, and basic performance for TPs.

### Multi-Trans Schedule Type

The multi-trans schedule type causes a transaction program to remain active between inbound conversation requests, with its resources available. The first inbound request starts the multi-trans TP. Subsequent requests can use the same instance of the transaction program and avoid the overhead of repeated resource allocation and deallocation.

Multi-trans processing is appropriate only for certain types of transaction programs. As a general rule, when properly implemented, multi-trans processing is appropriate for transaction programs that are requested often by multiple users, that have an initial high resource overhead, and that finish processing comparatively quickly.

For more information about multi-trans processing, see [“Multi-Trans Processing”](#) on page 30 and [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

## Multi-Trans Processing

---

A multi-trans program is typically coded with a multi-trans **shell**, an environment that performs initialization and termination processing, surrounding the part of the TP that holds conversations.

Table 1 on page 31 shows, at a very high level, how a multi-trans TP might work in an electronic mail application. It also shows how processing costs (resource responsibility) are divided between the multi-trans shell and individual users.

Table 1. Example of Multi-trans Resource Processing	
Action	Resource Responsibility
1. John is the first person to arrive in the morning, and requests his electronic mail.	none
2. John's request starts the system's mail log facility, a multi-trans TP named MLF.	shell
3. MLF performs initialization processing, and loads the notes for all users from permanent storage to a data space.	shell
4. MLF processes John's request and sends John his notes.	John
5. The APPC/MVS transaction scheduler causes MLF to wait.	shell
6. Janet requests her mail. MLF immediately sends Janet her notes.	Janet
7. A new note comes in for John, and MLF is signalled.	shell
8. The new note is read from permanent storage to MLF's data space. John is notified of the new note.	shell
9. John requests his new mail. MLF immediately sends John his new note.	John
10. MLF ends, and performs clean-up processing.	shell

When the multi-trans program is scheduled in response to an inbound allocate request, the multi-trans shell gets control first and allocates general resources. It then calls for the initial inbound request with the Get\_Transaction (ATBGTRN) callable service. When the conversation ends, the shell regains control and uses the Get\_Transaction call again to get the next conversation request on the queue.

Figure 18 on page 31 gives an overview of initialization and termination processing in a multi-trans TP.

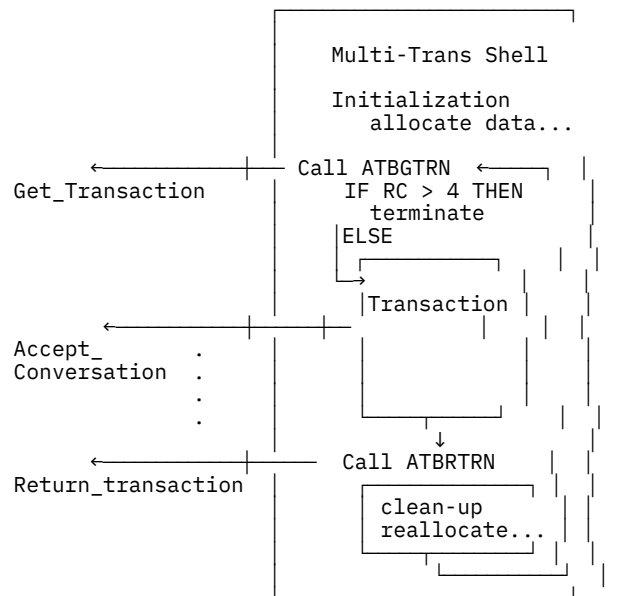


Figure 18. Initialization and Termination in Multi-trans Processing

To use multi-trans scheduling, a TP must define in its TP profile both a schedule type of multi-trans, and a special user ID (generic ID) for multi-trans shell processing.

The generic ID contained in the TP profile entry identifies the environment for the multi-trans shell. The shell runs under this generic ID during initialization, while it allocates general resources for the TP to use. The generic ID remains in effect until the first successful Get\_Transaction call, when the environment is personalized to the user ID associated with the inbound request. That personalized environment

covers the entire conversation and remains in effect until the next `Get_Transaction` call, or until the shell explicitly returns to its generic environment, typically to perform cleanup or data set reallocations between conversations. To return to its generic user ID, the shell can call the `Return_Transaction` (ATBRTRN) service.

The multi-trans shell can process an initial inbound `Allocate` request without first issuing `Get_Transaction`; in this case, the generic ID— not the user ID associated with the inbound request— identifies the conversation being processed. Using the multi-trans shell's generic ID this way can be useful when:

- A trusted, remote partner TP cannot supply a user ID on its inbound `Allocate` requests, or
- The installation wants the work that an APPC/MVS TP processes on behalf of its partners to run under only one user ID, rather than under several individual IDs.

## SMF Accounting of Multi-Trans Resources

SMF records can account for each phase of multi-trans processing. An installation can set up account numbers for each user of a multi-trans TP and differentiate between resources used for each user and resources used during shell processing.

For more information about SMF records in APPC, see [“Using SMF to Audit APPC Work” on page 210](#) and *z/OS MVS System Management Facilities (SMF)*.

## Security for Multi-Trans TPs

Multi-trans programs should be trusted applications. They must do whatever cleanup is necessary between transaction programs to ensure that resources are released and programs are isolated from one another and from any resources used exclusively by the shell.

Except for the cleanup responsibilities, multi-trans scheduling provides the same security protection as standard scheduling (checking user IDs, passwords, and profiles passed on each inbound conversation request). Each conversation with a multi-trans program runs under a personalized security environment, based on the user ID associated with the inbound request, when the multi-trans shell issues `Get_Transaction` and `Return_Transaction` to process conversations.

Because the generic ID covers processing that typically must be isolated from the different conversation partners, the generic ID must be secure from unauthorized specification or modification. To protect the multi-trans TP profile, which contains the generic ID, you can use RACF to control read and update access to the TP profile where the generic ID is specified. See [“Protecting Multi-Trans TP Profiles” on page 153](#).

## SYSOUT Processing for Multi-Trans TPs

A multi-trans TP typically must manage user-specific resources such as SYSOUT data separately for each conversation. If you want a multi-trans TP to process SYSOUT data for each user, you need to allocate a SYSOUT data set and explicitly free it for each conversation. If SYSOUT data sets are not deallocated or freed, SYSOUT data is not processed for users of the multi-trans TP until the entire TP and its shell environment are terminated.

For SYSOUT recommendations, see [“Specific Scheduler JCL Information for TP Profiles” on page 67](#).

## Assigning Multi-Trans TPs to their own Class

For performance reasons, IBM recommends that each transaction program scheduled as multi-trans be assigned to a unique class of APPC/MVS transaction initiators. Those classes are defined in `SYS1.PARMLIB` member `ASCHPMxx` and assigned to transaction programs in the TP profile. Each class consists of a range (maximum and minimum number) of transaction initiators that are available for running transaction programs of that class.

## Establishing a Multi-Trans TP that is Always Available

Multi-trans scheduling is especially suitable for transaction programs that must always be available to handle inbound requests. To ensure that an initiator is always available to run a multi-trans TP, a system programmer can do the following:

1. In the TP profile, assign the TP to a unique class
2. In the TP profile JCL, set the TIME parameter to NOLIMIT to prevent the IEFUTL exit of SMF from receiving control and terminating the address space that the multi-trans TP is running in.

Note that if a multi-trans TP receives no inbound requests for five minutes, the multi-trans TP receives a return code of 8 or 28 from Get\_Transaction. The multi-trans TP must either call the Get\_Transaction service again to wait for more work to arrive (if the return code is 8) or end the multi-trans TP (if the return code is 28). For more information about these return codes, see [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

To end or replace such a multi-trans TP, a system programmer can:

1. Set its TP profile inactive to prevent new inbound requests
2. Allow processing of queued requests to finish
3. Activate the TP profile of the new multi-trans TP.

## Logging Transaction Program Processing

---

Logging for transaction program (TP) processing is different from logging for other MVS work such as batch jobs, because TP processing is done in two phases. Batch jobs go through the converter/interpreter every time they execute, and all errors go to the job log. In contrast, TP processing separates the converter/interpreter phase from the execution phase and places errors from each phase in different logs.

### Converter/Interpreter Phase

When the APPC administration utility adds and modifies a TP profile, it checks the TP profile and its JCL for syntax errors. Output from this phase consists of TP profile syntax error messages, utility processing messages, and JCL conversion statements. Logging for messages from this phase is controlled by the SYSPRINT DD statement for the utility, as shown in the highlighted SYSPRINT DD statement in the following example of adding a TP profile. (For more information about the APPC administration utility, see [Chapter 6, “Using the APPC/MVS Administration Utility,”](#) on page 73.)

### Execution Phase

When a TP executes, the TP runtime messages, such as allocation and termination messages, go to a log named in its TP profile. Logging for messages from this second phase is defined in two places: an ASCHPMxx parmlib member and the TP profile. An ASCHPMxx parmlib member defines general message characteristics, and the TP profile defines specific log data set characteristics, as shown in the three highlighted parameters in the following example of adding a TP profile.

```

...
//STEP      EXEC  PGM=ATBSDFMU
//SYSPRINT DD   SYSOUT=A
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSSDOUT DD   DSN=UTILITY.OUTPUT,
//           DISP=(NEW,CATLG),
//           UNIT=SYSDA,VOL=SER=ICF003,
//           DCB=(RECFM=FB,LRECL=133,BLKSIZE=133),
//           SPACE=(TRK,(5,1))
//SYSIN      DD   DATA,DLM=XX
TPADD
  TPNAME(RUN_TEST)
  USERID(IBMUSER)
  ACTIVE(YES)
  TPSCHED_DELIMITER(##)
  TAILOR_SYSOUT(YES)
  TAILOR_ACCOUNT(YES)
  CLASS(A)
  TPSCHED_TYPE(STANDARD)
  JCL_DELIMITER(END_OF_JCL)
//TEST1     JOB   ,IBMUSER
//STEP      EXEC  PGM=TESTRUN
END_OF_JCL
  KEEP_MESSAGE_LOG(ALWAYS)
  MESSAGE_DATA_SET(&SYSUID;&SYSWUID;TESTLOG)
  DATASET_STATUS(NEW)
##
XX

```

Figure 19. Example of Adding a TP Profile

If your installation uses SMS, you may also specify parameters in the TP profile that control the allocation of an SMS-managed data set for a TP message log. Even if you do not specify those parameters, the TP message log might automatically be an SMS-managed data set anyway, depending on your installation's defaults.

TPs are not scheduled by a job entry subsystem, so the TP message log is not a system data set, such as JESJCL and JESYSMSG, and no such system data sets are created for logged TP information. To access the TP message log, use ISPF options rather than the mechanisms for controlling system data sets.

## The TP Message Log

The TP message log can be used for recovery and problem determination when an error occurs while a TP is processing. You can control the TP message log through parameters from the TP profile and from the APPC/MVS transaction scheduler parmlib member (ASCHPMxx). Table 2 on page 34 shows the parameters used in the TP message log definition, where the parameters are specified, and where to find detailed information about coding values for these parameters.

Table 2. Parameters Used in TP Message Log Definition		
Parameter	Where Specified	Coding Details in:
KEEP_MESSAGE_LOG	TP profile	<a href="#">“Transaction Scheduler Section” on page 61</a>
MESSAGE_DATA_SET	TP profile	<a href="#">“Transaction Scheduler Section” on page 61</a>
DATASET_STATUS	TP profile	<a href="#">“Transaction Scheduler Section” on page 61</a>
MSGLIMIT	ASCHPMxx	<a href="#">z/OS MVS Initialization and Tuning Reference</a>
MSGLEVEL	TP profile JCL or ASCHPMxx	<a href="#">z/OS MVS JCL Reference</a> or <a href="#">z/OS MVS Initialization and Tuning Reference</a>



## Deciding Which TP Message Log Values to Use

The default values for TP message log parameters tell APPC/MVS to create a new message log for each TP instance that ends on error. Depending on the environment in which the TP will run, and the type of TP, you might specify different values for the TP message log.

For example, the defaults might be appropriate for a TP running in a production system; however, you could bypass the message log completely, to reduce I/O and storage use. If the TP encounters a problem, you can change the parameter values only when you attempt to re-create the problem and collect diagnostic information. In contrast, when the TP is running in a test environment, you can use the values that allow you to collect all messages for all instances of the TP, whether or not each instance ends successfully.

Table 3 on page 35 lists the results of each possible combination of TP message log parameters. As you read through them to determine which combination best suits your needs, keep these facts in mind:

- Writing all messages for each instance of a TP requires more I/O than writing messages only on error, or bypassing the log.
- Having a new message log for each instance of a TP requires more storage than adding messages to a cumulative log, or reusing an existing log. The value of DATASET\_STATUS determines whether the log is new, cumulative, or reused; this parameter and its possible values are discussed in more detail in [“Selecting the Type of Message Log” on page 36](#).
- Logging messages for a standard TP differs from logging messages for a multi-trans TP. The differences mainly affect which value you choose for MSGLIMIT, which is discussed in more detail in [“Choosing a Value for the MSGLIMIT Parameter” on page 37](#).

<i>Table 3. Combination of Parameters Used in TP Message Log Definition</i>					
	<b>KEEP_MESSAGE_LOG</b>	<b>MESSAGE_DATASET</b>	<b>DATASET_STATUS</b>	<b>MSGLIMIT</b>	<b>MSGLEVEL</b>
<b>Bypass TP Message Log Method 1</b>	NEVER	N/A	N/A	N/A	N/A
<b>Bypass TP Message Log Method 2</b>	N/A	N/A	N/A	0	N/A
<b>Create New Log Each Time</b>	ALWAYS	&SYSUID. &SYSWUID. &TPDATE. &TPTIME. JOBLOG	NEW	0 - 15,000	1,1 (all msgs.)
<b>Create New Log Each Time (on error only)</b>	ERROR	&SYSUID. &SYSWUID. &TPDATE. &TPTIME. JOBLOG	NEW	0 - 15,000	1,0 (msgs. on error)
<b>Create Cumulative Log</b>	ALWAYS	<i>data set name</i>	MOD	0 - 15,000	1,1 (all msgs.)
<b>Create Cumulative Log (on error only)</b>	ERROR	<i>data set name</i>	MOD	0 - 15,000	1,0 (msgs. on error)
<b>Log Most Recent TP Instance</b>	ALWAYS	<i>data set name</i>	OLD	0 - 15,000	1,1 (all msgs.)

Table 3. Combination of Parameters Used in TP Message Log Definition (continued)					
	KEEP_ MESSAGE_LOG	MESSAGE_ DATASET	DATASET_ STATUS	MSGLIMIT	MSGLEVEL
<b>Log Most Recent TP Instance (on error only)</b>	ERROR	<i>data set name</i>	OLD	0 - 15,000	1,0 (msgs. on error)

## Selecting the Type of Message Log

If you choose to do anything other than bypass the message log, your choice of value for the DATASET\_STATUS parameter determines the type of message log:

### Value

#### Type of TP Message Log

#### NEW

A new log for each TP instance

#### OLD

A log for only the most recent TP instance

#### MOD

A cumulative log

Depending on the type of message log you want to use, you either allow APPC/MVS to allocate the data set, or allocate it yourself. If you allocate it yourself, use a record length of 133. (Each message written to the data set is one 133-byte record; messages are either padded with blanks or truncated to fit.)

- For a new log for each TP instance (DATASET\_STATUS is NEW):

In this case, APPC/MVS allocates a data set for you, specifying only primary space. The MSGLIMIT value for the class determines how many messages are logged for each TP instance, and APPC/MVS allocates a data set large enough to contain all those messages.

When you specify NEW, the data set is cataloged and kept once it is created. If you also specified ALWAYS for KEEP\_MESSAGE\_LOG, you might accumulate an inordinate number of data sets, because a new one is created for each instance of a TP.

- For a log of only the most recent TP instance (DATASET\_STATUS is OLD):

If you specify a DATASET\_STATUS of OLD, you must pre-allocate the data set. Estimate the amount of space required based on your knowledge of the TP's processing. You may specify both primary and secondary space. If you specify secondary space, the system allocates it, up to the extent limit for the type of data set, when the specified primary space is insufficient for the number of messages logged. The MSGLIMIT value for the class determines how many messages are collected in storage for the TP instance; the size of the data set, and any additional space the system can allow, determine how many of those messages are written to the data set.

To make sure you obtain all pertinent diagnostic information, you might have to adjust the specified amounts of primary and secondary space. If you don't have enough space allocated, a X'x37' abend results when the system attempts to write messages to the data set.

- For a cumulative log (DATASET\_STATUS is MOD):

If you specify a DATASET\_STATUS of MOD, IBM recommends that you pre-allocate the data set. If you do not, various installation or system defaults might result in the allocation of a data set that is not large enough for the amount of messages you want to collect.

Estimate the primary and secondary space required for a cumulative log by considering the following factors:

- How many messages might be generated by one instance of the TP.
- How many instances of the TP might run before you need to view the message log.

- The MSGLIMIT value specified for the class in which the TPs run. Remember that this value applies for each instance of the TP. For example, if the MSGLIMIT for the class is 50 messages, a cumulative log contains one set of no more than 50 messages for each TP instance.

Just as you may for a data set with a value of OLD, you may specify both primary and secondary space. If you specify secondary space, the system allocates it, up to the extent limit for the type of data set, when the specified primary space is insufficient for the number of messages logged.

To make sure you obtain all pertinent diagnostic information, you might have to adjust the specified amounts of primary and secondary space. If you don't have enough space allocated, a X'x37' abend results when the system attempts to write messages to the data set.

For multi-trans TPs only, APPC/MVS allows messages to wrap when the MSGLIMIT value is exceeded, so you can obtain more recent messages for each TP instance, at the cost of overwriting older messages. For more information about the MSGLIMIT value, and how it applies for standard versus multi-trans TPs, see [“Choosing a Value for the MSGLIMIT Parameter”](#) on page 37.

## Choosing a Value for the MSGLIMIT Parameter

The value of the MSGLIMIT parameter determines the maximum number of messages the system writes into the TP message log **for each instance of a TP** that runs within the class. Choosing a value for MSGLIMIT might be a trial-and-error process, because other factors also affect how many messages can be written: the type of TPs in the class; TP execution environment and processing; and the disposition and sometimes the size of the data set.

To calculate a MSGLIMIT value, consider starting with the default of 500 messages, and increasing that number when:

- The TP is a multi-trans TP or a long-running TP.
- The TP runs in a test, rather than a production, environment.
- The TP message log is cumulative (that is, the value of the DATASET\_STATUS parameter is MOD).

For standard TPs, APPC/MVS does not allow messages to wrap in the TP message log; if the MSGLIMIT value is exceeded, message ASB082I is written to the log, and message logging stops. In this case, to make sure you collect all the messages you need, increase the MSGLIMIT value and re-run the TP.

For multi-trans TPs, APPC/MVS does allow message wrapping, but only when the system has successfully written all the messages that are generated while the TP first runs under its shell (that is, the messages generated before the TP issues its first Get\_Transaction call). Once these shell messages have been written, the system continues writing messages until reaching the MSGLIMIT value; at this point, the system allows messages wrapping as follows:

1. The system writes ASB080I to the TP message log, placing it immediately after the last shell message. The system does not overwrite messages generated while the TP was first running under its shell.
2. Immediately after message ASB080I, the system overwrites existing messages with new messages until one of the following occurs:
  - The system reaches the MSGLIMIT value again. In this case, APPC/MVS repeats the wrapping process, beginning with overwriting the message that immediately follows ASB080I. APPC/MVS does not indicate how many times message wrapping occurs.
  - The multi-trans TP stops processing. Message ASB081I marks where wrapping stopped in the TP message log.

Although message wrapping might be convenient, it does not necessarily result in adequate diagnostic information; if the MSGLIMIT value is set too low, the messages you need could have been overwritten several times. Increasing the MSGLIMIT value increases the probability that valuable diagnostic information will not be overwritten.

If the MSGLIMIT value is changed while multi-trans TPs are running, the changed message limit does not affect the multi-trans programs until their environments are brought down and then restarted.

Figure 20 on page 38 contains a sequence of diagrams that illustrate how message wrapping can occur in the TP message log for a multi-trans TP. In this example, message wrapping occurs only once.

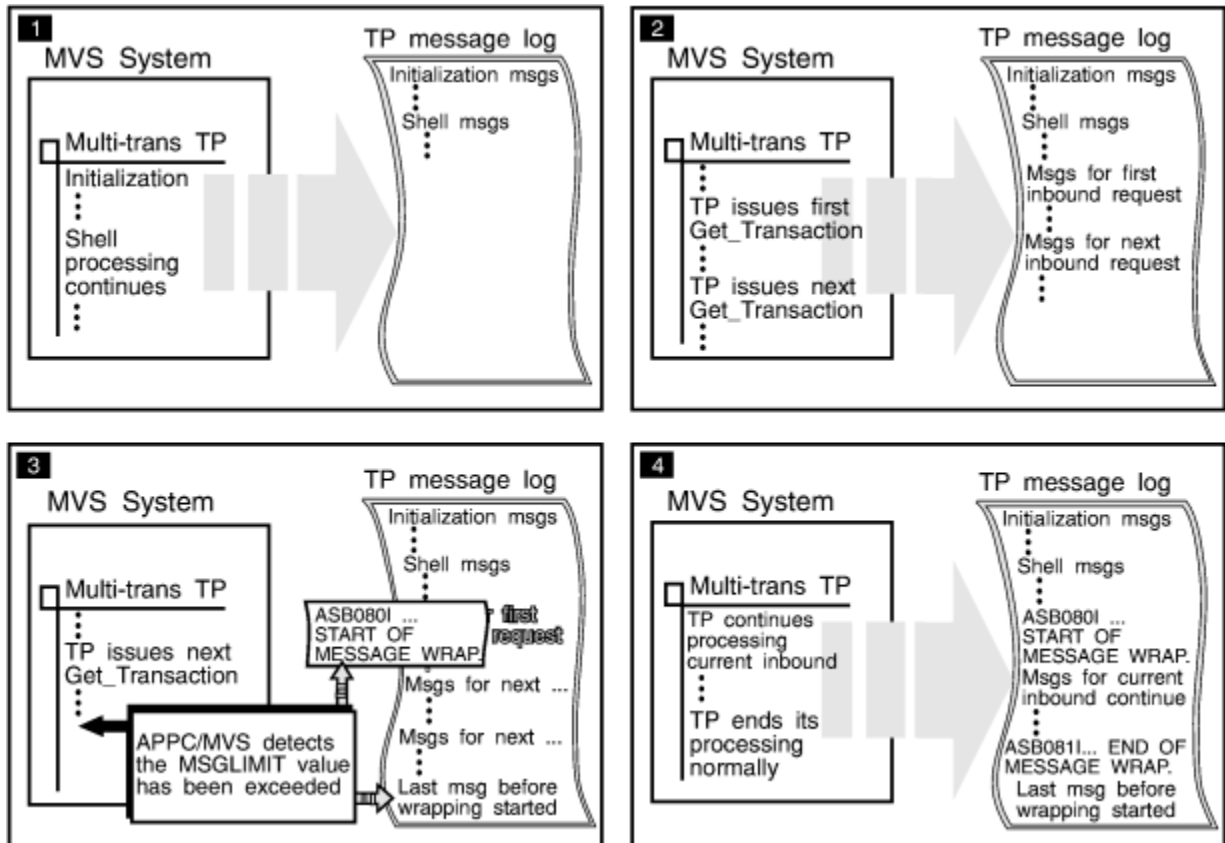


Figure 20. Message Wrapping in a Multi-Trans TP Message Log

Figure 21 on page 39 contains a sequence of diagrams that illustrate how messages might appear in a cumulative TP message log for the multi-trans TP in Figure 20 on page 38. In this example, the TP has run twice.

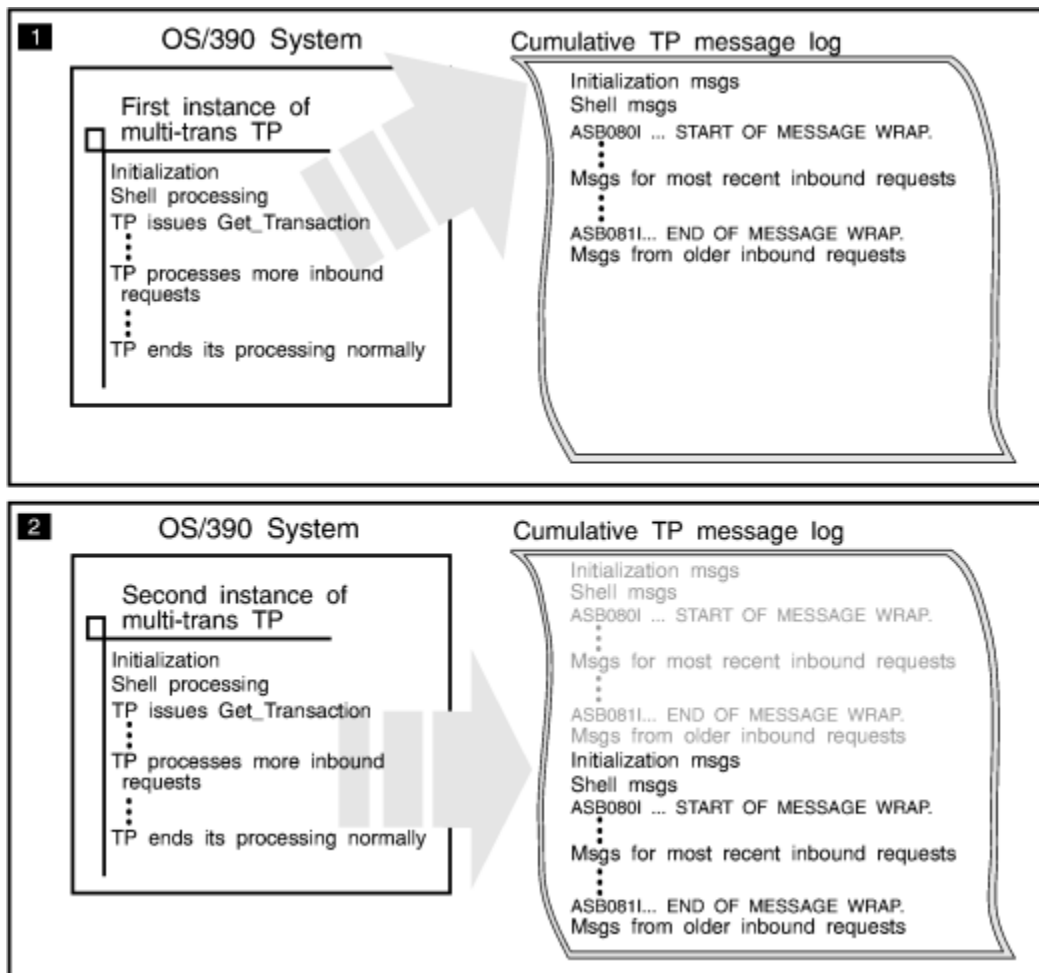


Figure 21. Messages in a Cumulative TP Message Log for a Multi-Trans TP

## Example of Using TP Message Log Parameters

To use a cumulative log for all messages with a limit of 10,000 messages, code the values for the highlighted parameters as shown in the following examples of a TP profile and an ASCHPMxx parmlib member.

```

TPNAME(RUN_TEST)
USERID(IBMUSER)
ACTIVE(YES)
TPSCHED_DELIMITER(##)
TAILOR_SYSOUT(YES)
TAILOR_ACCOUNT(YES)
CLASS(A)
TPSCHED_TYPE(STANDARD)
JCL_DELIMITER(END_OF_JCL)
//TEST1 JOB ,IBMUSER
//STEP EXEC PGM=TESTRUN
END_OF_JCL
KEEP_MESSAGE_LOG(ALWAYS)
MESSAGE_DATA_SET(TEST1.TESTLOG)
DATASET_STATUS(MOD)
##

```

Figure 22. TP Profile Parameters

```

CLASSADD
  CLASSNAME(A),
  MAX(10),
  MIN(2),
  RESPGOAL(.01),
  MSGLIMIT(10000)

TPDEFAULT
  REGION(256),
  TIME(1,30),
  MSGLEVEL(1,1),
  OUTCLASS(J)

```

Figure 23. ASCHPMxx Parameters

This example is for a test TP that creates a TP message log that contains all runtime messages. After a TP is successfully tested, an installation might change the KEEP\_MESSAGE\_LOG value to ERROR or NEVER to enhance TP runtime performance. Writing messages to the TP message log will affect CPU time and the response time of the TP.

## Viewing the TP Message Log During TP Run Time

When you set up the TP message log with the parameters described in the previous sections, you can access the log only after a TP stops running. If you need to access the TP message log between job steps, you can invoke program ASBSCHWL (write log routine) after a job step in the TP profile JCL. This routine enables you to view messages for the previous job step.

The following example invokes program ASBSCHWL in STEP2 and specifies that error messages, if any, go to data set IBMUSER.JOBLOG.DATASET.

```

TPNAME(RUN_TEST)
USERID(IBMUSER)
ACTIVE(YES)
TPSCHED_DELIMITER(##)
TAILOR_SYSOUT(YES)
TAILOR_ACCOUNT(YES)
CLASS(A)
TPSCHED_TYPE(STANDARD)
JCL_DELIMITER(END_OF_JCL)
//TEST      JOB      ,IBMUSER
//STEP1     EXEC  PGM=TEST1
//STEP2     EXEC  PGM=ASBSCHWL, PARM='JOBLOG'
//JOBLOG    DD      DSN=IBMUSER.JOBLOG.DATASET, DISP=SHR...
//STEP3     EXEC  PGM=TEST2
END_OF_JCL
##

```

Figure 24. Invoking the Write Log Routine

**Note:** If the PARM parameter is omitted in STEP2, JOBLGDCB is the default joblog DD name.

Write log routines can write messages only if the TP profile parameter KEEP\_MESSAGE\_LOG has a value of ALWAYS or ERROR. When KEEP\_MESSAGE\_LOG has a value of NEVER, no messages are written to any TP message log.

In the previous example, the write log routine controls only the TP messages written in STEP1. Otherwise, the following TP profile defaults apply:

```

KEEP_MESSAGE_LOG(ERROR)
MESSAGE_DATA_SET(&SYSUID;&SYSWUID;&TPDATE;&TPTIME;JOBLOG)
DATASET_STATUS(NEW)

```

Assuming that the ASCHPMxx parmlib member also uses defaults, the following parmlib member defaults apply:

```
MSGLIMIT(500)  
MSGLEVEL(1,0)
```

Given these defaults, if the TP abnormally ended in STEP3, messages would be written to the default TP message log data set (&SYSUID.&SYSWUID.&TPDATE.&TPTIME.JOBLOG), which can be accessed when the TP stops running. &SYSUID resolves to the user ID that invoked the TP, &SYSWUID resolves to the work unit identifier for the TP, and &TPDATE and &TPTIME resolve to the date and time the TP ran. For a multitrans TP, &SYSUID resolves to the generic userid. For a standard TP, &SYSUID resolves to the userid passed in with the inbound allocate request, unless security\_none is used (no userid passed), and then &SYSUID resolves to SYSUID.





---

## Chapter 4. Defining Scheduling Characteristics with ASCHPMxx

The APPC/MVS transaction scheduler initiates and schedules transaction programs in response to inbound requests for conversations. The ASCHPMxx member of the parmlib concatenation controls the function of the APPC/MVS transaction scheduler. Values in the parmlib member define and modify classes, and supply default TP characteristics when they are missing from the scheduler JCL section of the TP profile.

References:

[\*z/OS MVS Initialization and Tuning Reference\*](#)

[\*z/OS MVS System Messages, Vol 3 \(ASB-BPX\)\*](#)

[\*z/OS MVS Planning: Operations\*](#)

---

### ASCHPMxx Parmlib Member

The ASCHPMxx parmlib member primarily contains scheduling information for the APPC/MVS transaction scheduler using four statement types. The statement types are:

#### **CLASSADD**

Defines a class of transaction initiators to the APPC/MVS transaction scheduler configuration.

#### **CLASSDEL**

Deletes a class of transaction initiators from the APPC/MVS transaction scheduler configuration.

#### **OPTIONS**

Defines the default class for the APPC/MVS transaction scheduler and the default subsystem for starting transaction initiators.

#### **TPDEFAULT**

Supplies default scheduling characteristics when scheduling information is missing from a TP profile.

### Changing Values

An installation can change its scheduling characteristics using different versions of the ASCHPMxx parmlib member. For example, one parmlib member might contain setup values, while others contain statements that add new classes of transaction initiators (CLASSADD) or delete previous classes (CLASSDEL). The parmlib member may also re-specify statements with new parameter values to modify previous statements.

**Attention:** When modifying previous statements by respecifying them, the parmlib statements have a cumulative effect, and any one parmlib member might not contain the current scheduling information. If you use the CANCEL command to terminate the ASCH address space, you must respecify each parmlib member in its former order to reconstruct the previous scheduling definitions.

When parmlib statements are initially specified, omitted optional parameters receive default values. When parmlib statements are re-specified, however, omitted parameters in the CLASSADD statement assume the defaults, but omitted parameters in the OPTIONS and TPDEFAULT statements do not assume the defaults.

For example, a class is defined with specific characteristics by a CLASSADD statement. When another CLASSADD statement identifies the same class name but omits the MAX and MIN parameters, MAX assumes the default of 1 and MIN assumes the default of 0.

Therefore, to guarantee that intended values are not overridden by default values, re-specify all keywords on modifying statements.

## Using Default Values

The following sample parmlib member ASCHPM00 contains the default definitions for a single APPC/MVS transaction initiator class.

```
CLASSADD
  CLASSNAME(GENERAL)
  MAX(1)
  MIN(0)
  RESPGOAL(1)
  MSGLIMIT(500)

OPTIONS
  DEFAULT(GENERAL)

TPDEFAULT
  REGION(2M)
  TIME(1440)
  MSGLEVEL(1,0)
  OUTCLASS(A)
```

Figure 25. ASCHPM00

When you take the defaults, you get the following results:

### CLASSADD

The defaults are:

- The name of the class of transaction initiators is GENERAL.
- The maximum number of transaction initiators allowed for this class is one.
- No transaction initiators will be started for this class when the ASCH address space starts.
- The response time goal for TPs running within this class is one second. This is system response time, not user response time. For information about the RESPGOAL parameter, see [z/OS MVS Initialization and Tuning Reference](#).
- A maximum of 500 messages can be written to the TP message log, each time a TP within the class runs.

### OPTIONS

The defaults are:

- The name of the class in which to run a TP when no class name is specified in the TP profile is GENERAL.

**Note:** Because the SUBSYS parameter is omitted from the parmlib statement, all newly created APPC/MVS transaction initiators are started under the primary subsystem as defined in IEFSSNxx.

### TPDEFAULT

The defaults are:

- The region size assigned to TPs that do not specify a region size in their TP profile is 2M.
- There is no time limit assigned to TPs that do not specify a time limit in their TP profile.
- All statements and messages issued during TP profile add and modify processing will be generated. Statements and messages issued when a TP profile is accessed to run a TP will only be generated if the TP abnormally terminates.
- A is the class used as a default MSGCLASS for TPs whose profiles do not specify the MSGCLASS keyword in their JOB statements. (When the SYSOUT keyword does not include a specific output class, the value of MSGCLASS can be used as a default. Thus, OUTCLASS can affect how SYSOUT is processed.)

## Planning Specific Values

---

The values in the ASCHPMxx parmlib members control APPC/MVS scheduling characteristics. This section provides guidance for tasks involving installing and customizing the APPC/MVS transaction scheduler. The tasks described in this chapter include:

- Defining a class
- Modifying a class
- Deleting a class
- Defining default class options
- Defining default scheduling options.

## Defining a Class – CLASSADD Statement

---

The CLASSADD statement defines a class of transaction initiators to the APPC/MVS transaction scheduler. CLASSADD defines the following characteristics for a class:

**CLASSNAME(*name*)**

Class name

**MAX(*number*)**

Maximum number of transaction initiators

**MIN(*number*)**

Minimum number of transaction initiators

**RESPGOAL(*time*)**

Response time goal for each TP in the class

**MSGLIMIT(*number*)**

Maximum number of messages for the TP message log

At least one class definition is required for work that is assigned to the APPC/MVS transaction scheduler. If work comes in for a class that is not defined, the work is rejected.

Your installation should define classes based on the characteristics of the transaction programs that will run in a particular class. Programs in a class should have similar characteristics such as run-time, priority, schedule-type, and security. For example, you might define a class of transaction initiators for transaction programs that require SYSOUT processing. IBM recommends that each multi-trans program have its own class.

**Note:** If your installation uses the APPC administration dialog to maintain TP profiles, add class names to the dialog non-display panel ICQASE00 to keep the class list current. For more information, see [“Customizing the Dialog”](#) on page 90.

## Example of defining a class

The following example shows three CLASSADD statements in a parmlib member named ASCHPM1A. Each of the three classes runs different types of TPs. The class named FAST is for TPs with a response time goal of under .01 seconds, while the class named TEST is for testing new TPs before putting them on the production system. MULTI is a special class for a single, continuously running multi-trans TP.

```

CLASSADD
  CLASSNAME (FAST)
  MAX (10)
  MIN (2)
  RESPGOAL (.01)
  MSGLIMIT (200)

CLASSADD
  CLASSNAME (TEST)
  MAX (5)
  MIN (1)
  RESPGOAL (.5)
  MSGLIMIT (300)

CLASSADD
  CLASSNAME (MULTI)
  MAX (1)
  MIN (1)
  RESPGOAL (.1)
  /* MSGLIMIT will default to 500 */

```

*Figure 26. ASCHPM1A*

To activate the APPC/MVS transaction scheduler and these three classes, issue the START command as follows:

```
START ASCH,SUB=MSTR,ASCH=1A
```

If the APPC/MVS transaction scheduler was previously activated, issue the SET command to add the three classes, as follows:

```
SET ASCH=1A
```

To view the status of all classes, issue the DISPLAY ASCH command as follows:

```
DISPLAY ASCH,ALL
```

```

ASB101I 10.35.07 ASCH DISPLAY 209
CLASSES ACTIVE TRANS QUEUED TRANS IDLE INITS TOTAL INITS
00003 00005 00001 00000 00005
REGION TIME MSGLEVEL OUTCLASS SUBSYS
0002M 0001,30 1,0 J JES2
CLASS=FAST STATUS=ACTIVE ACTIVE TRANS=00003 MIN =00002
RESPGOAL=0.010000 QUEUED TRANS=00001 MAX=00010
DEFAULT=NO IDLE INITS=00000
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000005 ASID=0355
TPST=STANDARD USERID=JOE QT=*NONE*
JOBNAME=PAYROLL
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000004 ASID=6A22
TPST=STANDARD USERID=MARY QT=*NONE*
JOBNAME=PAYROLL
LTPN=MANAGE STATUS=ACTIVE WUID=A0000003 ASID=0412
TPST=STANDARD USERID=JOHN QT=*NONE*
JOBNAME=MANAGE5
LTPN=SECURITY STATUS=QUEUED WUID=A0000002 ASID=0012
TPST=STANDARD USERID=SECURE QT=000.001S
JOBNAME=*NONE*
CLASS=TEST STATUS=ACTIVE ACTIVE TRANS=00001 MIN =00001
RESPGOAL=0.500000 QUEUED TRANS=00000 MAX=00005
DEFAULT=NO IDLE INITS=00000
LTPN=FORMAT STATUS=ACTIVE WUID=A0000029 ASID=0477
TPST=STANDARD USERID=BARRY QT=*NONE*
JOBNAME=FORMAT
CLASS=MULTI STATUS=ACTIVE ACTIVE TRANS=00001 MIN =00001
RESPGOAL=0.100000 QUEUED TRANS=00000 MAX=00001
DEFAULT=NO IDLE INITS=00000
LTPN=MAIL STATUS=ACTIVE WUID=A0000001 ASID=0018
TPST=MULTITRANS USERID=DEPT5A QT=*NONE*
JOBNAME=DEPTMAIL

```

Figure 27. DISPLAY command output

## Modifying a Class – CLASSADD Statement

You can modify any characteristics of an existing class by overriding a previous CLASSADD statement with another CLASSADD statement that names the existing class and changes the parameters to be modified. When more than one CLASSADD statement exists for the same class, the most recently processed statement is in effect.

You can modify the following characteristics of a class:

### **CLASSNAME(name)**

Class name

### **MAX(number)**

Maximum number of transaction initiators

### **MIN(number)**

Minimum number of transaction initiators

### **RESPGOAL(time)**

Response time goal for each TP in the class

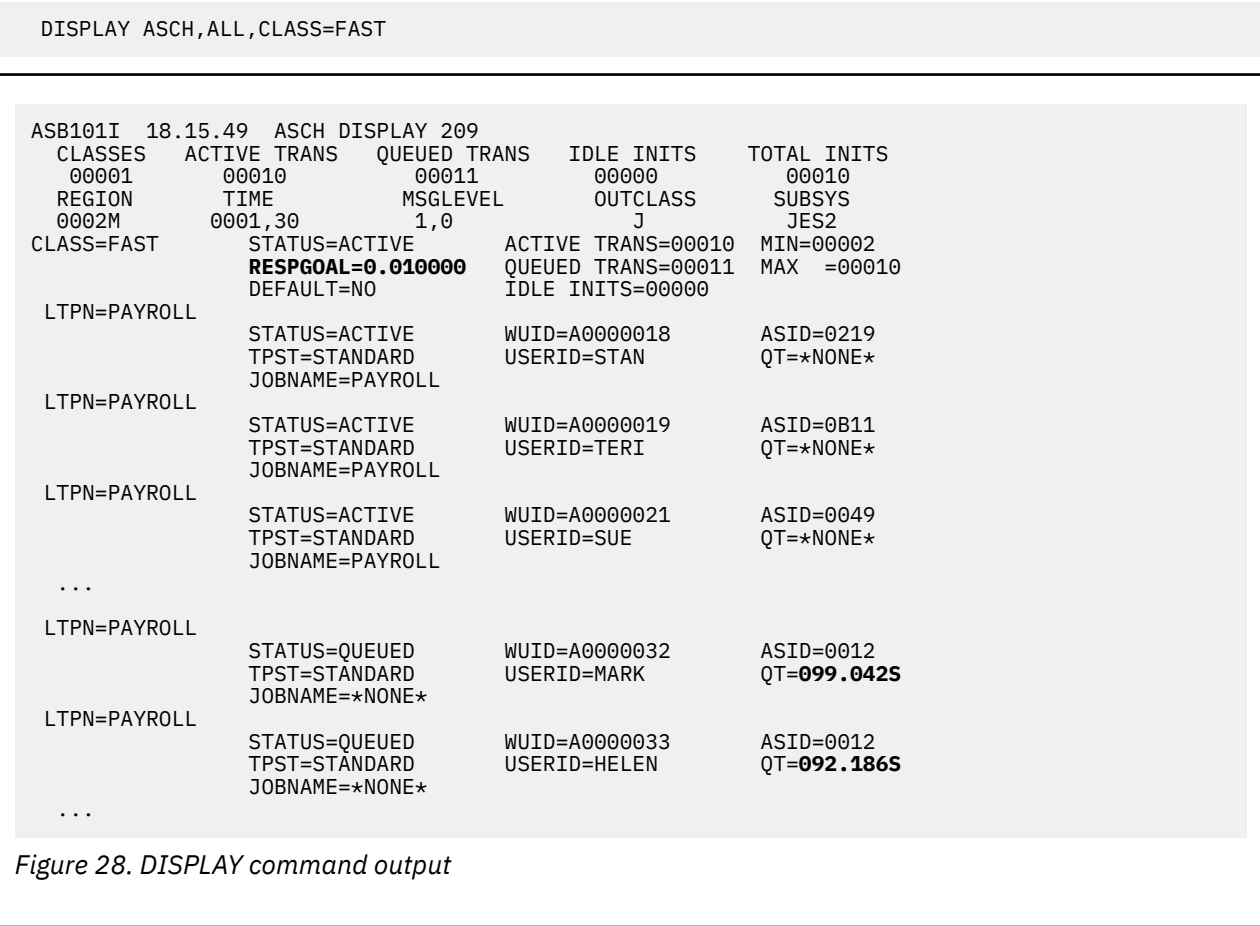
### **MSGLIMIT(number)**

Maximum number of messages for the TP message log

Remember that omitted parameters in the CLASSADD statement assume the defaults, which will override previously defined values. Therefore, to guarantee that intended values are not overridden by default values, re-specify all keywords on modifying statements.

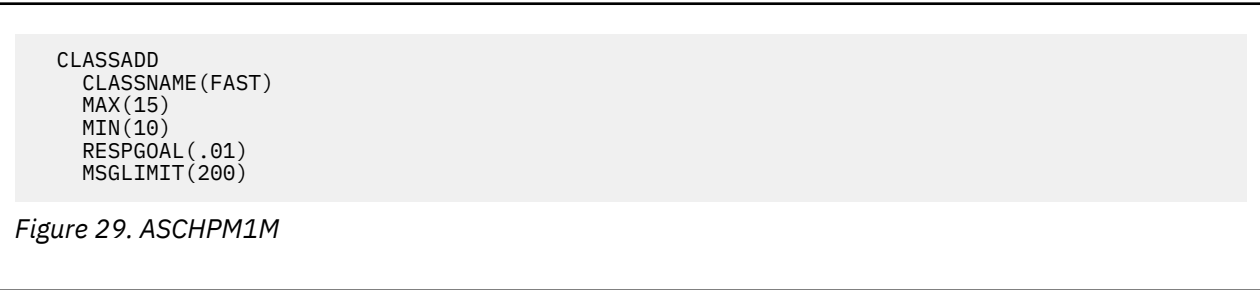
# Example of modifying a class

To determine whether a class is meeting its response time goal, issue the DISPLAY ASCH command with a specific class name.



In the previous example, class FAST has 11 queued transaction programs under the QUEUED TRANS heading. Two transaction programs have queue times (highlighted in the example) that greatly exceed the response time goal. A response time goal includes both queue time and run time, but, because the maximum number of initiators are running, the TPs on the queue cannot run.

To better meet the response time goal for the class, create a parmlib member that increases both the maximum and minimum number of transaction initiators, as in the following example.



Activate the parmlib member with the following SET command.

```

SET ASCH=1M

```

# Deleting a Class – CLASSDEL

The CLASSDEL statement deletes a class of transaction initiators from the APPC/MVS transaction scheduler. One CLASSDEL statement must be defined for each class that is deleted.

The CLASSDEL statement contains:

**CLASSNAME(name)**

Identifies the class

**WORKQ(DRAIN|PURGE)**

Specifies whether the system drains or purges the work queue

When a CLASSDEL statement is processed, work is either drained or purged, depending on the option that you specified for the WORKQ keyword. If you specified the DRAIN option, work that is currently queued for this class is allowed to complete. The system does not delete the class until all work has been processed. If you specified the PURGE option, however, work that is currently queued for this class is rejected. Only work that is already running in an initiator is processed. If you do not specify either DRAIN or PURGE, the DRAIN default is used.

**Note:** If your installation uses the APPC administration dialog to maintain TP profiles, make sure permanently deleted class names are also deleted from the dialog non-display panel ICQASE00 to keep the class list current. For more information, see [“Customizing the Dialog” on page 90](#).

## Example of deleting a class

To delete a class (for example, a test class that is no longer needed), code a parmlib member with the CLASSDEL statement. In the following example, the test class work queue will be purged.

```
CLASSDEL
  CLASSNAME(TEST)
  WORKQ(PURGE)
```

Figure 30. ASCHPM1D

To activate the parmlib member and delete the class, issue this SET command.

```
SET ASCH=1D
```

If you want to see the results of the CLASSDEL, issue the DISPLAY command as follows:

```
DISPLAY ASCH,ALL,CLASS=TEST
```

As long as TPs are running in the class, information about the class is displayed; however, the status of the class changes from ACTIVE to TERMINATING.

```
...
CLASS=TEST      STATUS=TERMINATING  ACTIVE TRANS=00001  MIN=00001
                 RESPGOAL=0.500000  QUEUED TRANS=00000  MAX=00005
                 DEFAULT=NO        IDLE INITS=00000
LTPN=FORMAT     STATUS=ACTIVE        WUID=A0000029      ASID=0477
                 TPST=STANDARD      USERID=BARRY       QT= *NONE*
```

Figure 31. DISPLAY command output

## Defining Default Options – OPTIONS

The OPTIONS statement defines class defaults for the APPC/MVS transaction scheduler:

**DEFAULT(name)**

Identifies default class

**SUBSYS(name)**

Identifies default subsystem that does SYSOUT processing and the subsystem under which APPC/MVS transaction initiators are started

You can modify defaults by overriding a previous OPTIONS statement with another OPTIONS statement that changes parameter values. When more than one OPTIONS statement exists, the most recently processed parameter values are in effect.

### Example of Defining a Default Class

To set up a default class for TPs that do not specify a class in their TP profiles, code a parmlib member with the OPTIONS statement and specify the default class you want to use. In the following example, class SLOW is added and then defined as the default class. Because no SUBSYS parameter is specified, this parmlib member uses the primary JES subsystem as the default.

```
CLASSADD
  CLASSNAME(SLOW)
  MAX(5)
  MIN(0)
  RESPGOAL(60)
  MSGLIMIT(300)

OPTIONS
  DEFAULT(SLOW)
```

Figure 32. ASCHPM2A

To activate the default class, issue the SET command as follows:

```
SET ASCH=2A
```

To change the default class or subsystem, create another parmlib member with an OPTIONS statement that names another default class or subsystem, and issue the SET command.

## Defining Default Scheduler Options – TPDEFAULT

The TPDEFAULT statement supplies default scheduling information when it is missing from the scheduler JCL section of a TP profile. These scheduling defaults apply only to TPs scheduled by the APPC/MVS transaction scheduler. TP profile defaults defined in TPDEFAULT include the following:

**REGION(size)**

Amount of virtual storage needed for the TP

**TIME(time)**

CPU time limit for running the TP

**MSGLEVEL(level,level)**

Level of messages generated

**OUTCLASS(name)**

Default class for MSGCLASS

To change the default values for TPs, create another parmlib member with a TPDEFAULT statement, and specify new values for the keyword parameters.



## Example of Defining Scheduling Defaults

To provide defaults for TP profiles that do not specify certain scheduling parameters, code a parmlib member that contains a TPDEFAULT statement.

```
TPDEFAULT
REGION(2M)
TIME(1,30)
MSGLEVEL(1,0)
OUTCLASS(J)
```

Figure 33. ASCHPM3A

This TPDEFAULT example specifies that:

- The amount of virtual storage assigned to the TP is 2M.
- TPs that take longer than 1 minute 30 seconds of CPU time will be abnormally terminated.
- All statements and messages issued during TP profile add and modify processing will be generated, but statements and messages issued when a TP profile is accessed to run a TP will be generated only if the TP abnormally terminates.
- J is the class used as a default MSGCLASS for TPs whose profiles do not specify the MSGCLASS keyword in their JOB statements. (When the SYSOUT keyword does not include a specific output class, the value of MSGCLASS can be used as a default. Thus, OUTCLASS can affect how SYSOUT is processed.)

To activate these values in member ASCHPM3A, issue the SET command as follows:

```
SET ASCH=3A
```

Scheduling defaults for the system can be viewed by issuing the following DISPLAY command:

```
DISPLAY ASCH,SUMMARY
```

ASB101I	21.28.12	ASCH	DISPLAY		
CLASSES	ACTIVE	TRANS	QUEUED	TRANS	IDLE INITS
00003	00010		00011		00000
REGION	TIME		MSGLEVEL		OUTCLASS
0002M	0001,30		1,0		J
					TOTAL INITS
					00010
					SUBSYS
					JES2

Figure 34. DISPLAY command output

## Examples using ASCHPMxx Parmlib members

Because of the cumulative way the ASCHPMxx parmlib members interact, you might consider creating a separate member for:

- Initial setup
- Each anticipated modification
- Deletion of each class

For example, you could create the following parmlib member for initial setup:

```

CLASSADD
  CLASSNAME (FAST)
  MAX (10)
  MIN (2)
  RESPGOAL (.01)
  MSGLIMIT (200)

CLASSADD
  CLASSNAME (SLOW)
  MAX (5)
  MIN (0)
  RESPGOAL (60)
  MSGLIMIT (300)

CLASSADD
  CLASSNAME (MULTI)
  MAX (1)
  MIN (1)
  RESPGOAL (.1)
  MSGLIMIT (500)

OPTIONS
  DEFAULT (SLOW)
  SUBSYS (JES2)

TPDEFAULT
  REGION (2M)
  TIME (1,30)
  MSGLEVEL (1,0)
  OUTCLASS (J)

```

*Figure 35. ASCHPM1S*

Along with the initial setup, you might consider creating parmlib members for anticipated modifications such as:

- Changing the default class for varying system workloads
- Modifying the MAX and MIN parameters of a class for tuning purposes.

For example, you could create the following parmlib member for changing the default class from FAST to SLOW for second shift:

```

OPTIONS
  DEFAULT (SLOW)

```

*Figure 36. ASCHPM2M*

You could also create the following parmlib member for increasing the MAX and MIN values for class FAST, and use it to tune scheduler performance:

```

CLASSADD
  CLASSNAME (FAST)
  MAX (15)
  MIN (10)
  RESPGOAL (.01)
  MSGLIMIT (200)

```

*Figure 37. ASCHPM1M*

You might also want to create parmlib members for deleting each class. The following parmlib members delete each of the classes shown in the CLASSADD statements listed above.

```
CLASSDEL  
CLASSNAME (FAST)
```

*Figure 38. ASCHPM1D*

```
CLASSDEL  
CLASSNAME (SLOW)  
WORKQ (PURGE)
```

*Figure 39. ASCHPM2D*

```
CLASSDEL  
CLASSNAME (MULTI)  
WORKQ (DRAIN)
```

*Figure 40. ASCHPM3D*

## Tracking Changes in Scheduling Definitions

There are two ways to keep track of changes to the ASCHPMxx parmlib member:

- Keep a hardcopy log of every ASCHPMxx member that was activated by using the LIST option on the START ASCH and SET ASCH commands.
- View the current scheduling configuration by issuing the DISPLAY ASCH,ALL command.

### Keeping a Hardcopy Log

You can define, on the HARDCOPY statement of a CONSOLxx parmlib member, a hardcopy log that provides a permanent record of ASCHPMxx parmlib activity. For information about defining the hardcopy log, see [z/OS MVS Planning: Operations](#).

To list the contents of each activated parmlib member to the operator console and to the hardcopy log, include the LIST option on the START and SET commands. For example, when starting ASCH using parmlib member ASCHPM1S, issue the START command as follows:

```
START ASCH, SUB=MSTR, ASCH=(1S, L)
```

When changing the configuration with parmlib member ASCHPM1M and ASCHPM3D, issue the SET command with the LIST option as follows:

```
SET ASCH=(1M, 3D, L)
```

This command displays the contents of both ASCHPM1M and ASCHPM3D on the console screen and stores the information in the hardcopy log.

```

ASB038I ASCHPM1M : CLASSADD
ASB038I ASCHPM1M : CLASSNAME(FAST)
ASB038I ASCHPM1M : MAX(15)
ASB038I ASCHPM1M : MIN(10)
ASB038I ASCHPM1M : RESPGOAL(.01)
ASB038I ASCHPM1M : MSGLIMIT(200)

ASB038I ASCHPM3D : CLASSDEL
ASB038I ASCHPM3D : CLASSNAME(MULTI)
ASB038I ASCHPM3D : WORKQ(DRAIN)

```

Figure 41. SET command LIST option output

## Viewing the current scheduling configuration

A way to get a "snapshot" of the scheduling configuration is with the DISPLAY command. To view the classes and their workload, issue the DISPLAY command as follows:

```
DISPLAY ASCH,ALL
```

```

ASB101I 09.22.81 ASCH DISPLAY 209
CLASSES ACTIVE TRANS QUEUED TRANS IDLE INITS TOTAL INITS
00002 00005 00000 00000 00005 00005
REGION TIME MSGLEVEL OUTCLASS SUBSYS
0002M 0001,30 1,0 J JES2
CLASS=FAST STATUS=ACTIVE ACTIVE TRANS=00004 MIN=00002
RESPGOAL=0.010000 QUEUED TRANS=00000 MAX=00010
IDLE INITS=00000
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000018 ASID=0219
TPST=STANDARD USERID=STAN QT=*NONE*
JOBNAME=PAYROLL
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000019 ASID=0B11
TPST=STANDARD USERID=TERI QT=*NONE*
JOBNAME=PAYROLL
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000021 ASID=0049
TPST=STANDARD USERID=SUE QT=*NONE*
JOBNAME=PAYROLL
LTPN=PAYROLL STATUS=ACTIVE WUID=A0000032 ASID=2568
TPST=STANDARD USERID=MARK QT=*NONE*
JOBNAME=PAYROLL
CLASS=SLOW STATUS=ACTIVE ACTIVE TRANS=00001 MIN=00000
RESPGOAL=0060.000 QUEUED TRANS=00000 MAX=00005
DEFAULT=YES IDLE INITS=00000
LTPN=BATCH5 STATUS=ACTIVE WUID=A0000033 ASID=0012
TPST=STANDARD USERID=IBMUSER QT=*NONE*
JOBNAME=BATCH5

```

Figure 42. DISPLAY command output

---

## Chapter 5. Controlling the Execution of Transaction Programs

Two types of administrative data, a TP profile and side information, help control the flow of conversations in an APPC/MVS network. A TP profile contains the scheduling and security information that might be necessary to run a TP in MVS. Side information contains the translation of a symbolic destination name used by an MVS local TP when issuing an outbound allocate request. (APPC/MVS servers can also specify symbolic destination names when registering to receive inbound conversations.)

Both types of administrative data are stored in VSAM key sequenced data sets (KSDS), with at least one VSAM file for TP profiles and only one for side information. The APPC administration utility (ATBSDFMU) maintains the TP profile and side information files; you submit batch jobs that can add, modify, retrieve, and delete entries. An interactive panel dialog version of the APPC/MVS administration utility is available.

For information about the utility, see [Chapter 6, “Using the APPC/MVS Administration Utility,”](#) on page 73, and for information about the dialog, see [Chapter 7, “Using the APPC/MVS Administration Dialog,”](#) on page 83.

This chapter discusses the following aspects of controlling the execution of TPs:

References:

[\*z/OS DFSMS Access Method Services Commands\*](#)  
[\*z/OS MVS JCL Reference\*](#)  
[\*z/OS MVS Installation Exits\*](#)

---

### Determining Scheduling Characteristics

One of the main purposes of a TP profile is to describe the environment necessary to schedule and run the TP. This information usually comes from the scheduling section of the profile. When scheduling information is missing from a TP profile and the TP is scheduled with the APPC/MVS transaction scheduler, the OPTIONS and TPDEFAULT statements in an ASCHPMxx parmlib member can provide some defaults, such as a scheduling class. However, for proper scheduling of TPs running under the APPC/MVS transaction scheduler, give consideration to each parameter in the scheduling section of the profile. Each of these parameters is explained in detail in [“Transaction Scheduler Section”](#) on page 61.

Transaction programs on MVS can be scheduled by the APPC/MVS transaction scheduler (ASCH) or by an installation-defined scheduler. How you define the scheduling portion of a TP profile depends on the transaction scheduler used. Before you create profiles, ask the person who determines scheduling policy for the following information:

- For TPs scheduled by the APPC/MVS transaction scheduler:
  - The class name to use for each type of TP. (For example, IBM recommends that TPs scheduled as standard use a different class from those scheduled as multi-trans.)
  - Whether a default has been established for class in an OPTIONS statement of ASCHPMxx.
  - Whether default information was established in a TPDEFAULT statement of ASCHPMxx.
- For TPs scheduled by a scheduler other than the APPC/MVS transaction scheduler:
  - How the TP profile should be adapted to reflect scheduling characteristics of the transaction scheduler.
  - The name of the transaction scheduler exit used to syntax check the TP profile information.

# Defining the VSAM Key Sequenced Data Sets (KSDS)

The VSAM files that contain the TP profiles and side information must be defined before you can create TP profiles and side information. The number of files you need to define as well as the size of each file depends on several factors. These topics are discussed in the following sections.

For general information about VSAM files, see [z/OS DFSMS Access Method Services Commands](#).

The person who actually defines the VSAM files should be an experienced VSAM programmer familiar with the restrictions when migrating a VSAM file to another system, the requirements for changing the size of a pre-defined VSAM file, and the ability to view a VSAM KSDS online.

## Determining How Many Files to Define

All the side information used by a system must be contained in one file that is named in the SIDEINFO statement of an APPCPMxx parmlib member. TP profiles, however, can be contained in either a single file or in many files. Files for side information and TP profiles can be shared by more than one system. Using a global resource serialization star or ring complex is one method of allowing systems to safely share these files. See [z/OS DFSMS Using Data Sets](#) for more information about sharing VSAM data sets among systems.

The number of files you need to define for profiles is related to the number of local LUs in MVS. You can name one TP profile file for each local LU when the LU is created. You can name a different file for each LU, or name the same file for several or all LUs. For example, in Figure 43 on page 56, File-1 is the TP profile file for several LUs, namely LU-A, LU-B, and LU-C. File-2 is the exclusive TP profile file for LU-D.

LU-A }	File-1
LU-B }	
LU-C }	
LU-D }	File-2

Figure 43. Relationship of files to LUs

Whether you need many TP profile files or only one might depend on:

### Isolation

Although access to the individual TP profiles in a file can be controlled by level (system, group, and user), an installation might choose to isolate the communication that passes through a single LU. If so, the LU might need an exclusive TP profile file.

### Scheduling

An LU is associated with a single transaction scheduler, and the TP profile file for that LU should contain profiles for TPs scheduled by that transaction scheduler. In the previous figure, if LU-A, LU-B, and LU-C are associated with the APPC/MVS transaction scheduler, and LU-D is associated with transaction scheduler XYZ, File-1 should contain profiles for TPs scheduled by the APPC/MVS transaction scheduler and File-2 should contain profiles for TPs scheduled by XYZ.

### Testing

An installation might choose to devote an LU to pre-production testing and associate it with a separate TP profile file that contains only profiles for TPs to be tested.

### Delegation of administrative responsibilities

Files can be associated with departments or groups within an installation and maintained by different administrators.

### Sharing files on more than one system

When systems share files, the files cannot be updated when in use. Therefore, you might want to create two rotating copies of each file, one to use and the other to update. For more information, see [“Restrictions on Invoking the APPC/MVS Administration Utility” on page 80](#).

Basically, many files allow flexibility, but a single file is easier to administer. Carefully consider whether the multiple files you need for flexibility outweigh the efficiency of keeping all TP profiles together in one file.

# Determining the Size of Each File

Assuming that you separate side information entries from TP profile entries, and that TP profiles for different transaction schedulers are also separated, you could hypothetically define three types of files:

- Side information file (1 only)
- TP profile file for the APPC/MVS transaction scheduler (1 or more)
- TP profile file for a scheduler other than the APPC/MVS transaction scheduler (1 or more)

## Side Information File

When planning the size of the side information file, consider the size of each entry and the number of entries needed. The size of each side information entry does not vary. The side information key is always 112 bytes, and the remainder of the side information is 136 bytes, resulting in a total of 248 bytes for each entry. This number follows the RECORDSIZE keyword in the VSAM file definition.

The number of entries in a side information file depends on the number of partner LUs and the number of unique TP/logon mode combinations. Outbound requests can be made to many destinations and each combination of destination/TP/logon mode needs a separate side information entry and a unique side information key.

Figure 44 on page 57 shows how outbound requests for two TPs (TP1 and TP2) require five side information entries because of the various combinations of partner LU/TP/logon mode.

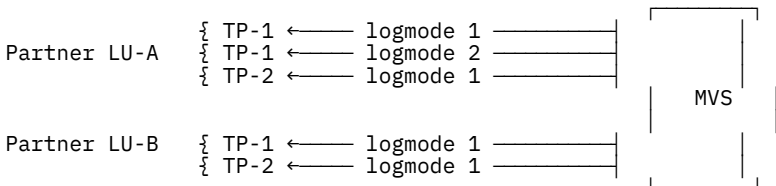
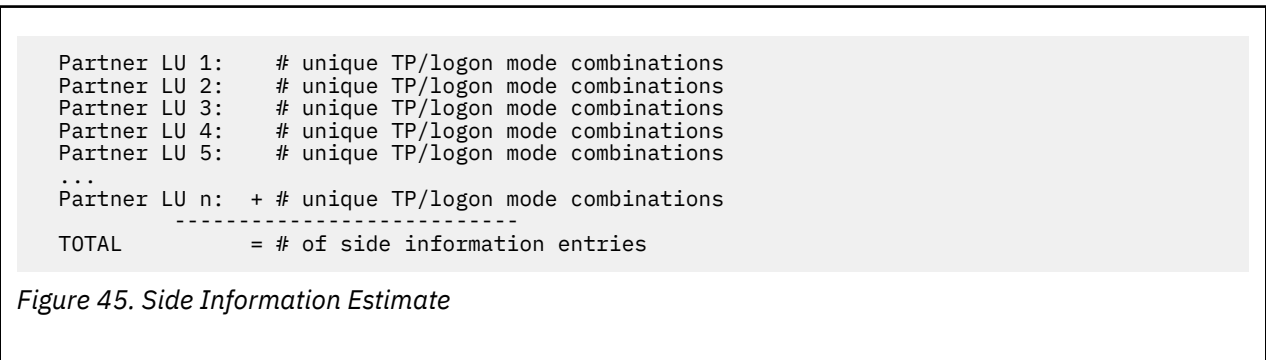


Figure 44. Combinations of Partner LUs, TPs, and Logon Modes

Therefore, to estimate the number of entries for a side information file, add together the number of unique TP/logon mode combinations for each partner LU.



The estimated number of entries is the first number following the RECORDS keyword in the VSAM file definition. The second number following the RECORDS keyword is the additional number of entries that can be added for expansion. For information about how much expansion to allow, see [z/OS DFSMS Access Method Services Commands](#).

SYS1.SAMPLIB member ATBSIVSM contains a sample VSAM definition for a side information file.

## TP Profile Files for the APPC/MVS Transaction Scheduler

When planning a TP profile file for TPs scheduled by the APPC transaction scheduler, consider the approximate size of each entry and the number of entries. The size of each entry varies depending on the size of the JCL portion of the profile. The TP profile non-JCL portion is 624 bytes, but the JCL portion can range from 2 records (160 bytes) to 100 records (8000 bytes).

Based on the previous numbers, you can estimate the size of an entry in a TP profile to be between 3824 bytes (non-JCL of 624 + average JCL of 3200) and 7024 bytes (non-JCL of 624 + maximum JCL of 6400). These numbers follow the RECORDSIZE keyword in the VSAM file definition.

The number of entries in a TP profile file depends on the number of TPs and the number of groups and users who have profiles for a TP. TPs can have a single system-level profile and any number of group- or user-level profiles.

In Figure 46 on page 58, LU-A and LU-B share the same TP profile file. The file contains four TP profiles for TP-1, two TP profiles for TP-2, and two TP profiles for TP-3.

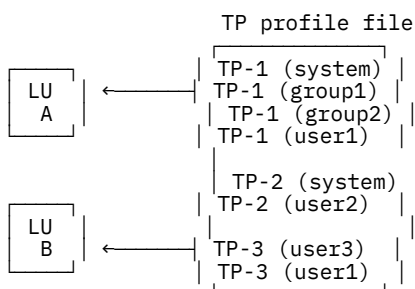


Figure 46. TP Levels in a File

Therefore, to estimate the number of entries for a TP profile file, add for each TP the number of group-level profiles, the number of user-level profiles and, if one exists, a system-level profile.

```

TP 1:  (# of groups) + (# of users) + (1 system, if any)
TP 2:  (# of groups) + (# of users) + (1 system, if any)
TP 3:  (# of groups) + (# of users) + (1 system, if any)
TP 4:  (# of groups) + (# of users) + (1 system, if any)
...
TP n:  + (# of groups) + (# of users) + (1 system, if any)
-----
TOTAL = # of TP profile entries

```

Figure 47. TP Profile Estimate

The estimated number of entries is the first number following the RECORDS keyword in the VSAM file definition. The second number following the RECORDS keyword is the additional number of entries that can be added for expansion. For information about how much expansion to allow, see [z/OS DFSMS Access Method Services Commands](#).

SYS1.SAMPLIB member ATBTPVSM contains a sample VSAM definition for a KSDS for TP profiles scheduled by the APPC/MVS transaction scheduler.

## TP Profile Files for Non-APPC/MVS Transaction Schedulers

When planning a TP profile file for TPs not scheduled by the APPC/MVS transaction scheduler, consider the approximate size of each entry and the number of entries. The size of each entry can vary depending on the information required by the transaction scheduler. The TP profile non-JCL information is fixed at 624 bytes.

To analyze the number of entries, follow the procedure for analyzing entries for an APPC/MVS transaction scheduler file as explained in the previous section.

## Using Database Tokens for File Security

As a security aid, you can assign a database token to each file. The database token is essentially a single variable that represents the file name in a security definition statement. For example, the database token used in combination with other information allows a specific security definition for each TP profile in the



file. Being able to single out individual TPs in a file permits a security administrator to assign security on a TP-by-TP basis.

For more information about securing TPs with a database token, see [“Controlling Access to Database Tokens”](#) on page 150.

## Creating a TP Profile

A TP profile contains identification, security, and scheduling information for a target TP that resides in MVS and is scheduled in response to an inbound allocate request. Every TP in MVS that receives an inbound allocate request and is scheduled by the APPC/MVS transaction scheduler must have a TP profile. (Note that inbound requests that are processed by APPC/MVS servers are not normally scheduled and therefore do not require the use of a TP profile.) Each TP profile consists of a TP profile key, a program attributes section, and a transaction scheduler section.

### TP Profile Key

The TP profile key uniquely identifies a TP profile within the VSAM file. The key consists of the TP name and a level.

Name of the TP —————→ TPNAME(name)  
Level of the TP —————→ SYSTEM|GROUPID(id)|USERID(id)

*Figure 48. TP Profile Key*

#### TPNAME(name)

Indicates one of the following:

- 1- through 64-character name of the transaction program. Valid characters are those from the 00640 character set and the Type A character set. For descriptions of these character sets, see [Appendix A, “Character Sets,”](#) on page 233. If the name will be used in the APPC/MVS administration dialog or in DISPLAY commands, do not use an asterisk (\*) in the name.
- 2- to 4-character name of the transaction program (if the transaction program is an SNA service TP). Although the names can be 2 to 4 characters, they are normally 4 characters in length. To specify the SNA service TPNAME in the APPC/MVS administration utility, you must map the 2- to 4-character SNA TPNAME into the following 7- to 9-character format:

→X'nn'yyy

where:

**nn**

is two digits that represent hex characters from 00 through 3F, excluding 0E and 0F. These two digits correspond to the first character in the SNA TP name.

**yyy**

is 1 to 3 Type A characters. A listing of Type A characters is in [Appendix A, “Character Sets,”](#) on page 233. These 1 to 3 characters correspond to the last 1 to 3 characters of the SNA TPNAME.

[Figure 49](#) on page 60 shows how the 4-character SNA TPNAME 37,C1,C2,C3 maps into the APPC/MVS administration utility TPNAME →X'37'ABC. Because the last 3 characters of the SNA TPNAME, C1,C2,C3, are TYPE A characters, you can convert them using the listing in [Appendix A, “Character Sets,”](#) on page 233. The first character in the SNA TPNAME, 37, cannot be displayed, however, so you must map the first character into the fourth and fifth characters shown in [Figure 49](#) on page 60.

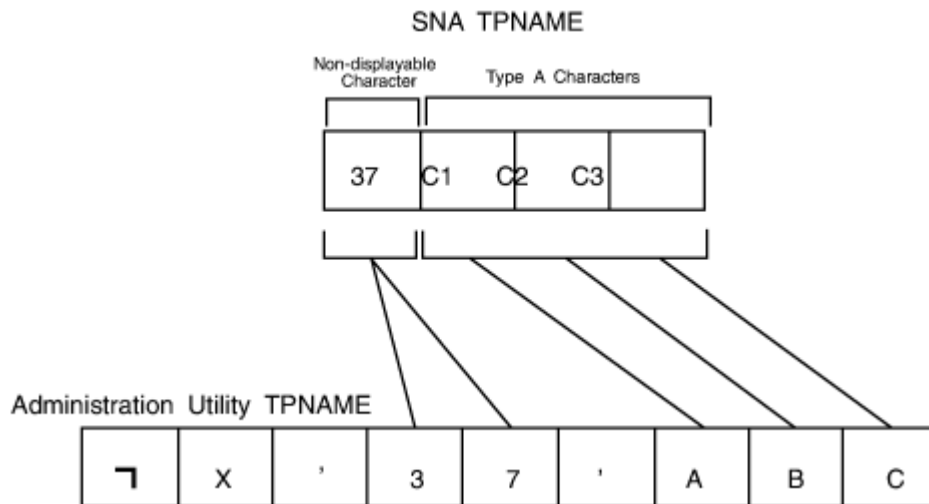


Figure 49. Mapping an SNA TPNAME into an Administration Utility TPNAME

### **SYSTEM|GROUPID(id)|USERID(id)**

Indicates the level of the TP profile. You can customize a TP's processing for different audiences by creating different profiles for the TP and making each profile available to a designated audience, such as all users defined to an LU, a group of users, or an individual user. You identify the audience for a TP by specifying a level in the TP profile key.

A single TP can have profiles for all three levels; one available for all users on the system (highest level), one for a specified group of users, and one for an individual user (lowest level). When APPC/MVS receives an incoming allocate request for a TP with more than one profile, it uses the TP profile with the lowest level to which the requestor has access.

When a TP issues an allocate request with a security specification of security\_none, no group or user ID is passed and only system-level TPs can be searched.

For information about profile access security, see “Controlling User Access to TP Profiles and Side Information on MVS” on page 148. For more information about specifying levels, see “Associating TPs and LUs with the Appropriate Level” on page 215.

## **Program Attributes Section**

The program attributes section of the TP profile contains information about the program's status and defines a delimiter to mark the beginning and the end of the transaction scheduler section.

```
Active Status —————→ ACTIVE(YES|NO)
Beginning of Scheduler Section → TPSCHED_DELIMITER(delimiter1)

      scheduler information

End of Scheduler Section —————→ delimiter1
```

Figure 50. Program Attributes Section

### **ACTIVE(YES|NO)**

Indicates whether the TP can be accessed for scheduling. When a TP profile is active, the TP can be scheduled by a transaction scheduler. When a TP profile is not active, the TP cannot be scheduled until its profile is changed back to active status. The default is YES, which indicates the TP is active.

The ACTIVE keyword allows an installation to temporarily deactivate a TP profile when the TP fails for an unknown reason. Deactivation prevents continued requests for the TP, which would result in repeated failures, and allows the programmer to read the message log for information about why the TP failed.

### TPSCHED\_DELIMITER(*delimiter1*)

Marks the beginning of the transaction scheduler section and identifies a delimiter that will be used to mark the end of the transaction scheduler section. The delimiter can be from 1 through 53 characters but cannot be a character string that appears within the transaction scheduler section itself. When the delimiter marks the end of the transaction scheduler section, it must appear on a line of its own and start in the first column.

## Transaction Scheduler Section

The transaction scheduler section contains information about how the program will be scheduled by the APPC/MVS transaction scheduler. Programs not scheduled by the APPC/MVS transaction scheduler must provide an exit to syntax check the format expected by that scheduler.

```
Utility command and exit —————> Command_name TPSCHED_EXIT(exit name)
... TP profile key ...
... Attributes section ...
Security SYSOUT Update —————>TAILOR_SYSOUT(NO|YES)
Security Account Update —————>TAILOR_ACCOUNT(NO|YES)
Scheduler Class Name —————>CLASS(class name)
TP Scheduler Type —————>TPSCHED_TYPE(STANDARD|MULTI_TRANS)
ID for Multi_Trans TPs —————>GENERIC_ID(generic userid)
Beginning of TP JCL —————>JCL_DELIMITER(delimiter2)
                                //jobname JOB ...
                                //stepname EXEC ...

End of TP JCL —————>delimiter2
Message Log Option —————>KEEP_MESSAGE_LOG(ERROR|ALWAYS|NEVER)
Name of Message Data Set —————> MESSAGE_DATA_SET(data set name)
Status of Message Data Set —————> DATASET_STATUS(NEW|OLD|MOD)
SMS Storage Class —————> STORAGE_CLASS(class name)
SMS Management Class —————> MANAGEMENT_CLASS(class name)
SMS Data Class —————> DATA_CLASS(class name)
```

Figure 51. APPC/MVS Transaction Scheduler Section

### TPSCHED\_EXIT(ASCH|*exit name*)

Names the exit that syntax checks the scheduler section. This keyword must appear on the same line as the utility command, after the command name when adding or modifying TP profiles. For example, when adding a TP profile that will use the ASCH scheduler exit, you can specify:

```
TPADD TPSCHED_EXIT(ASCH)
```

The default is ASCH, so this keyword can be omitted for TPs scheduled by the APPC/MVS transaction scheduler.

For information about writing an exit to check syntax, see [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#).

### TAILOR\_SYSOUT(NO|YES)

Indicates whether a program tailors each transaction's SYSOUT with additional SYSOUT information from the requestor's security profile. When the requestor uses a security specification of security\_none on the Allocate call, no SYSOUT tailoring can occur. The default is NO.

The SYSOUT specification in the TP profile JCL applies to all transaction instances of the program whether or not tailoring is requested. To guarantee that output is processed before a TP ends, include the FREE=CLOSE parameter with the SYSOUT specification. For SYSOUT recommendations, see [“Specific Scheduler JCL Information for TP Profiles”](#) on page 67.

When the RACF security product is used, the SYSOUT information is stored in the WORKATTR segment of the RACF user profile. For information about the RACF user profile, see [Chapter 10, “Setting up Network Security,”](#) on page 133.

**Note:** When SYSOUT tailoring is requested, OUTPUT statement keywords have varying default and override values. See individual OUTPUT keyword descriptions in [z/OS MVS JCL Reference](#) for specific defaults and overrides.

**TAILOR\_ACCOUNT(NO|YES)**

Indicates whether a program tailors each transaction's account with the account information from the requestor's security profile. If there is no account number in the requestor's security profile and account tailoring was requested, the account number passed is 00000000. When the requestor uses a security specification of security\_none on the Allocate call, no account tailoring can occur. The default is NO.

When no account tailoring is requested, the initial account specification in the TP profile JCL records applies to all transaction instances of the program.

When the RACF security product is used, the account information is stored in the WORKATTR segment of the RACF user profile. For information about the RACF user profile, see [Chapter 10, "Setting up Network Security,"](#) on page 133.

When a TP profile indicates that accounts are to be tailored, an installation can verify each transaction's account number with exit IEFUAV. For information about IEFUAV, see [z/OS MVS Installation Exits](#).

**Note:** When account tailoring is requested and account numbers appear in both the TP profile JCL and the security profile, the security profile's account number overrides the TP profile's account number.

**CLASS(class name)**

Names the 1- through 8-character scheduler class into which the TP is scheduled. Classes set up scheduling characteristics, such as the maximum and minimum number of transaction initiators for the class, and the response time for each transaction.

Classes are named by CLASSADD statements in ASCHPMxx parmlib members. If no class is specified in the TP profile, APPC/MVS administration searches for a default class named by the OPTIONS statement in an ASCHPMxx parmlib member. For information about the ASCHPMxx parmlib member, see [Chapter 4, "Defining Scheduling Characteristics with ASCHPMxx,"](#) on page 43.

**TPSCHED\_TYPE(STANDARD|MULTI\_TRANS)**

Indicates whether the TP is scheduled as standard or multi-trans. The default is standard.

When a TP is scheduled as standard, an environment for the transaction is created, resources are allocated for its use, and the TP is initialized in its isolated environment. When the transaction completes, the resources are cleaned up and the TP ends.

When a standard TP issues or receives multiple allocate calls, each call causes re-allocation of resources and re-initialization of the TP, resulting in a new instance of the TP.

To enhance performance for TPs that are frequently invoked, you can write a multi-trans TP and specify a schedule type of multi-trans. Multi-trans programs assume responsibility for resource cleanup between transaction requests so that they can remain initialized for subsequent requests.

**GENERIC\_ID(generic user ID)**

Names a generic user ID for a multi-trans program. Because the generic user ID covers processing that typically must be isolated from the different conversation partners, it must be secure from unauthorized specification or modification. To protect the generic user ID, you can use RACF or an equivalent security product to control read and update access to the TP profile where the generic user ID is specified. For more information, see ["Protecting Multi-Trans TP Profiles"](#) on page 153.

**JCL\_DELIMITER(delimiter2)**

Marks the beginning of the JCL that will actually schedule and attach the TP, and identifies a delimiter that will be used to mark the end of the JCL. The delimiter can be from 1 through 59 characters but cannot be a character string that appears within the JCL itself. When you use the delimiter to mark the end of the JCL, it must appear on a line of its own starting in the first column.

**JCL records**

The JCL by which the TP is scheduled and attached. The JCL allowed in this section is a subset of all JCL statements. Two statements are required:

- JOB statement, which is the place to specify limits. For tracking purposes, specify a unique jobname. If resources the TP uses are to be billed to an account that is not tailored, include an account number.

- EXEC statement, which names the program to invoke the TP.

To print the TP message log, invoke the write log routine ASBSCHWL from a second EXEC statement. For information about the TP message log, see [“Logging Transaction Program Processing” on page 33](#).

Note that DD statements, although not required by APPC/MVS, might be required by the program.

When certain job-related information is not specified, APPC/MVS administration searches for default information in a TPDEFAULT statement of an ASCHPMxx parmlib member. For information about TPDEFAULT, see [“Defining Default Scheduler Options — TPDEFAULT” on page 50](#).

Examples and restrictions appear in [“Specific Scheduler JCL Information for TP Profiles” on page 67](#).

### **KEEP\_MESSAGE\_LOG(ERROR|ALWAYS|NEVER)**

Indicates the conditions under which messages are written to the TP message log data set. If ERROR is specified, messages are written only when an error occurs. ALWAYS indicates that messages at all times are written, and NEVER indicates that no messages are written. The default is ERROR.

This keyword works together with a MSGLEVEL keyword parameter that can be specified in the scheduler JCL section of the TP profile. (A default MSGLEVEL parameter is provided in the TPDEFAULT statement of the ASCHPMxx parmlib member.) The MSGLEVEL keyword controls the generation of messages and KEEP\_MESSAGE\_LOG controls writing the messages to the TP message log.

### **MESSAGE\_DATA\_SET(data set name)**

Names the data set where TP messages are written. When no name is specified, the name defaults to &SYSUID.&SYSWUID.&TPDATE.&TPTIME.JOBLOG. The variables resolve to the following:

- For a multi-trans TP, &SYSUID resolves to the generic userid. For a standard TP, &SYSUID resolves to the userid passed in with the inbound allocate request, unless security\_none is used (no userid passed), and then &SYSUID resolves to SYSUID.
- &SYSWUID resolves to the work unit identifier
- &TPDATE resolves to the date that the TP ran. &TPDATE is in the form Dyyyyddd where yyyy is the year and ddd is the day of the year. The initial D simply identifies the qualifier as being a date.
- &TPTIME resolves to the time that the TP ran. &TPTIME is in the form Thhmmss where hh is the hour (of a 24-hour clock), mm is the minutes, and ss is the seconds. The initial T simply identifies the qualifier as being a time.

#### **Note:**

1. You can use &SYSUID, &SYSWUID, &TPDATE, and &TPTIME as variables for any qualifiers in the data set name.
2. When an inbound allocate request has the security specification of security\_none, no user ID is passed with the request, and the system variable &SYSUID resolves to SYSUID.

### **DATASET\_STATUS(NEW|OLD|MOD)**

Indicates the status of the TP message log data set. OLD indicates that a message log data set already exists and is cataloged. When messages are written to an old data set, they overwrite previous data. MOD indicates that a data set will be created and cataloged if it doesn't already exist. If it does exist, messages are added to the end of the data set. NEW indicates that the data set does not yet exist. The default is NEW.

**Note:** When the status of the TP message log is NEW, the disposition is CATALOG, which means the data set is kept once it is created. If a TP profile specifies KEEP\_MESSAGE\_LOG(ALWAYS), a new data set will be created each time the TP runs and message log data sets will accumulate.

### **STORAGE\_CLASS(class name)**

Names the 1- through 8-character storage class used to define a new SMS-managed message data set. If the message log data set is to be SMS-managed, this keyword is required. If not specified, dynamic allocation defaults are used to allocate the message data set.

**Note:** Use this keyword only when all of the following are true:

- The data set is new.
- SMS is available at your installation.
- You want to create a SMS-managed message data set.

#### **MANAGEMENT\_CLASS(class name)**

Names the 1- through 8-character management class for an SMS-managed message data set. For data sets already identified as SMS-managed (through a storage class name), SMS will provide a management class.

**Note:** Use this keyword only when all of the following are true:

- The data set is new.
- SMS is available at your installation.
- You want to create a SMS-managed message data set.

#### **DATA\_CLASS(class name)**

Names a 1- through 8-character data class for an SMS-managed message data set. The specified data class must have a record length of 133 bytes. For data sets already identified as SMS-managed (through a storage class name), SMS will provide a data class with a record length of 133 bytes.

**Note:** Use this keyword only when all of the following are true:

- The data set is new.
- SMS is available at your installation.
- You want to create a SMS-managed message data set.

## Summary of TP Profile Keywords

Table 4. Summary of TP Profile Keywords		
Keyword	Value(s)/Length	Default
TPNAME	1-64 Type 00640 characters or Type A characters or 2-4 characters (for an SNA service TP)	None
SYSTEM	N/A	
GROUPID	1-8 (subject to security product's group ID rules)	None
USERID	1-8 (subject to security product's user ID rules)	None
TPSCHED_EXIT	ASCH   1-8 Type A characters	ASCH
ACTIVE	YES   NO	YES
TPSCHED_DELIMITER	1-53 characters (can contain any character combination)	None
CLASS If omitted, you must define a default class in an ASCHPMxx parmlib member.	1-8 Type A characters	None
TAILOR_SYSOUT	NO   YES	NO
TAILOR_ACCOUNT	NO   YES	NO
TPSCHED_TYPE	STANDARD   MULTI_TRANS	STANDARD
GENERIC_ID	1-8 Type A characters	None

Table 4. Summary of TP Profile Keywords (continued)		
Keyword	Value(s)/Length	Default
KEEP_MESSAGE_LOG	ERROR   ALWAYS   NEVER	ERROR
MESSAGE_DATA_SET	1-44	&SYSUID.&SYSWUID.&TPDATE. &TPTIME.JOBLOG
DATASET_STATUS	NEW   OLD   MOD	NEW
STORAGE_CLASS	1-8 Type A characters	None
MANAGEMENT_CLASS	1-8 Type A characters	None
DATA_CLASS	1-8 Type A characters	None
JCL_DELIMITER	1-59 characters (can contain any character combination)	None

## Creating Side Information

Side information contains the translation of a symbolic destination name that can be used on outbound allocate requests, or by APPC/MVS servers, when registering to receive inbound conversations. Programs specify a symbolic destination name that represents the TP name, logon mode, and partner LU on the Allocate or Register\_for\_Allocates call.

Side information consists of the following sections:

Symbolic Destination Name —————→ DESTNAME(sym\_dest\_name)

*Figure 52. Side Information Key*

TP Name —————→ TPNAME(name)  
 Logon Mode Name —————→ MODENAME(mode)  
 Partner LU Name —————→ PARTNER\_LU(name)

*Figure 53. Side Information Data*

### **DESTNAME(sym\_dest\_name)**

Identifies the 1- through 8-character symbolic destination name for the partner TP.

### **TPNAME(name)**

Indicates one of the following:

- 1- through 64-character name of the transaction program. Valid characters are those from the 00640 character set and the Type A character set. For descriptions of these character sets, see Appendix A, “Character Sets,” on page 233. If the name will be used in the APPC/MVS administration dialog or in DISPLAY commands, do not use an asterisk (\*) in the name.
- 2- to 4-character name of the transaction program (if the transaction program is an SNA service TP). Although the names can be 2 to 4 characters, they are normally 4 characters in length. To specify the SNA service TPNAME in the APPC/MVS administration utility, you must map the 2- to 4-character SNA TPNAME into the following 7- to 9-character format:

```
~X'nn'yyy
```

where:

**nn**

is two digits that represent hex characters from 00 through 3F, excluding 0E and 0F. These two digits correspond to the first character in the SNA TP name.

yyy

is 1 to 3 Type A characters. A listing of Type A characters is in Appendix A, “Character Sets,” on page 233. These 1 to 3 characters correspond to the last 1 to 3 characters of the SNA TPNAME.

Figure 54 on page 66 shows how the 4-character SNA TPNAME 37,C1,C2,C3 maps into the APPC/MVS administration utility TPNAME →X'37'ABC. Because the last 3 characters of the SNA TPNAME, C1,C2,C3, are TYPE A characters, you can convert them using the listing in Appendix A, “Character Sets,” on page 233. The first character in the SNA TPNAME,37, is nondisplayable, however, so you must map the first character into the fourth and fifth characters shown in Figure 54 on page 66.

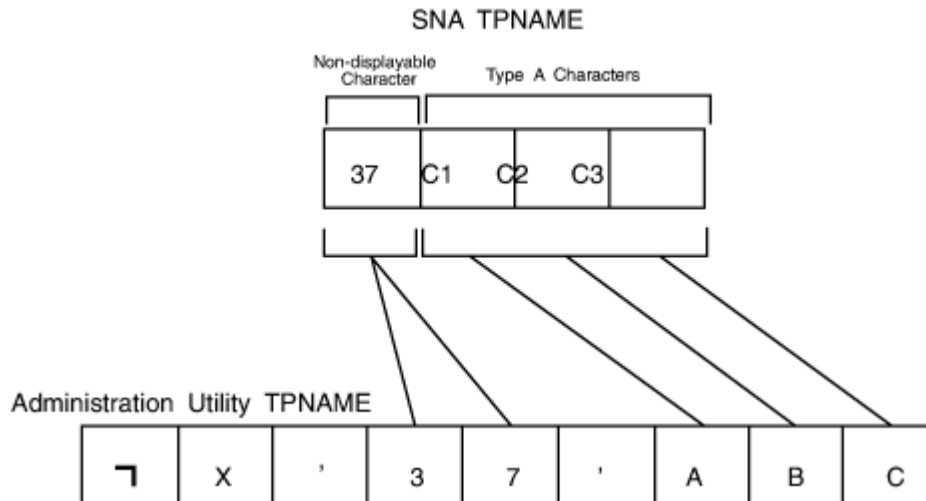


Figure 54. Mapping an SNA TPNAME into an Administration Utility TPNAME

#### MODENAME(mode)

Names the logon mode for the SNA session connecting the local LU with the partner LU. If no mode name is specified, a default mode name might be available. For information about when defaults are available, see “Specifying a Logon Mode for a Conversation” on page 111.

#### PARTNER\_LU(name)

Identifies the 1- through 17-character name of the partner LU where the partner TP resides. If no partner LU name is specified, APPC/MVS assumes the TP resides in the local LU.

This value can be one of the following:

- Network LU name only (1-8 byte Type A character string)
- A VTAM generic resource name (1-8 byte Type A character string). This value should not be a generic resource name if any APPC/MVS servers use the LU value from a side information entry for a Register\_for\_Allocates call. Register\_for\_Allocates accepts only a specific LU name.
- Combined network\_ID and network LU name (two 1-8 byte Type A character strings, concatenated by a period): *network\_ID.network\_LUname*. The network LU name can be a VTAM generic resource name.

## Example of Side Information

In the following COBOL example, the symbolic destination name USR3NEWS is used in a CPI Communications outbound Initialize\_Conversation call and is resolved by its side information in the example box.

```
MOVE "USR3NEWS" TO SYM-DEST-NAME.  
CALL "CMINIT" USING CONVERSATION-ID,  
                    SYM-DEST-NAME,  
                    CM-RETCODE.
```



```
DESTNAME(USR3NEWS)
MODENAME(MODE3)
TPNAME(NEWS)
PARTNER_LU(USER3LU)
```

Figure 55. Example of Side Information

## Summary of Side Information Keywords

Table 5. Summary of Side Information Keywords		
Keyword	Value(s)/Length	Default
DESTNAME	1-8 alphanumeric characters This value is not case sensitive.	None
TPNAME	1-64 Type 00640 characters or Type A characters or 2-4 characters (for an SNA service TP)	None
MODENAME	0-8 Type A characters	None
PARTNER_LU	0-17 Type A characters	None

## Defining TP Profiles and Side Information Early

When an installation must define a large number of TPs, it might be advantageous to define them early, before the installation has completely migrated to a new release of MVS. When the TP profiles and side information are defined early, the TPs are ready to run soon after the new release is installed.

Although the APPC/MVS administration utility (ATBSEDFMU) is available with MVS/ESA SP 4.2, it is possible to install it on a lower level system. The utility can run on MVS/ESA SP 3.1 or later. The parts that comprise the utility can be moved to the lower level system from a test system running MVS/ESA SP 4.2 or later.

When installed early, the utility can scan for errors and perform the services for all side information entries and for TP profile entries of TPs not scheduled by the APPC/MVS transaction scheduler. However, for TPs scheduled by the APPC/MVS transaction scheduler, the JCL portion of the TP profile cannot be checked for correct syntax. If the JCL contains an error, the TP will fail when invoked, and messages might not define the JCL error.

If you want to ensure that the JCL of a TP profile defined early has correct syntax before the TP is invoked, you can deactivate the TP profile when defining it, and activate it with a TPMODIFY. The JCL will then be checked for syntax errors. Activation of profiles is controlled by the ACTIVE keyword in the attributes section of the TP profile.

## Specific Scheduler JCL Information for TP Profiles

The JCL used in the scheduler section of a TP profile is a subset of all JCL statements. Two statements are required:

### JOB statement

The JOB statement (the first JCL statement) gives the program designer a place to specify limits for the TP (like maximum PAGES, LINES, BYTES and CARDS). For retrievability and debugging, use a unique jobname for each TP. If resources the TP uses will be billed to an account, specify an account number.

## EXEC statement

The EXEC statement names the program that will invoke the TP. If your installation is running JES2 Version 4.1 or later or JES3 Version 4.2.1 or later, you can add error control within the EXEC statement by an IF statement similar to the following:

```
//MAINTRAN EXEC PGM=APPCTRAN
//          IF (MAINTTRAN.RC > 8 OR ABEND=TRUE) THEN
//ERROR     EXEC PGM=NOTIFY
/* Inform somebody about the error
//PRNTLOG   EXEC PGM=ATBWTB
/* Print information to a log if error occurred
//MSGLOG    DD  SYSOUT=A,DEST=(NODE5.FRED)
/* Route output to a specific user
//          ENDIF
```

## SYSOUT Recommendations

TPs can use the same JES SYSOUT functions available to other MVS applications, with the exception that SYSOUT data sets allocated by TPs are treated as spin data sets. As with non-spin batch data sets, TP SYSOUT data sets are processed when they are unallocated. When FREE=CLOSE is coded in the SYSOUT specifications in the TP profile JCL, unallocation of the spin data set occurs immediately after it closes.

When a TP is scheduled as standard, its transaction initiator unallocates data set resources when the TP ends. When a TP is scheduled as multi-trans, it runs under a shell which, between requests, does not do the normal cleanup of a transaction initiator. Thus, the SYSOUT data set is not processed until all requests for the multi-trans have completed and the multi-trans ends. Multi-trans TPs that create output must close their SYSOUT data sets after each request completes. To guarantee processing of SYSOUT data, code FREE=CLOSE on TP profile SYSOUT specifications, or include the CLOSE macro with the FREE option within the TP.

To guarantee that SYSOUT data sets are processed before initiator cleanup, IBM recommends that all SYSOUT specifications, for both standard and multi-trans TPs, include the FREE=CLOSE parameter.

## JCL Size Restrictions

The maximum record limit for JCL within TP profiles is approximately 100 records, depending on the complexity of the records. This maximum record limit includes the records that are invoked through a JCLLIB statement. An example of a JCLLIB statement follows in [“PROCLIB Restrictions” on page 71](#).

## Unsupported Statements and Restrictions

You cannot use the following statements in the JCL scheduler portion of the TP profile:

- Job entry control language (JECL) statements. These are the statements associated with JES2 or JES3; for example:

```
//*FORMAT
//*MAIN
```

If JES2 JECL statements are coded, they are detected as JCL errors. If JES3 JECL statements are coded, they are ignored and appear as comments.

- Internal reader control statements; for example:

```
/*EOF
/*DEL
```

- Instream commands:

```
"//  command          "
"//  COMMAND '... command text ...' "
```

- The XMIT statement:

The XMIT JCL statement has no function in an APPC scheduling environment. If you code an XMIT statement, however, it must be syntactically correct to avoid JCL errors.

## Restrictions on &SYSUID Variable

- Using the variable &SYSUID anywhere in your JCL degrades performance.
- Using the variable &SYSUID on the JOB statement NOTIFY parameter might result in the failure of an Allocate request. If you code NOTIFY=&SYSUID and the user ID specified through the Allocate service is longer than seven characters, the Allocate request will fail. IBM recommends that you avoid using the NOTIFY parameter in an APPC scheduling environment.
- Using the &SYSUID variable on the DD statement SUBSYS parameter results in inconsistent symbolic substitution.
- Note that, as described in the following section, you cannot use &SYSUID in a data set name in the TP profile JCL if the requestor's security specification is security\_none.
- Data sets on a JCLLIB statement should not use the &SYSUID variable.

## Data Set Naming Restrictions

Data set names in the TP profile JCL that use the variable &SYSUID cannot be resolved when the requestor's security specification is security\_none. Because no user ID is passed on the allocate request, the request ends in error.

## JOB Statement Restrictions

- The JOB statement MSGLEVEL keyword has limited function in an APPC scheduling environment. See [“Logging Transaction Program Processing” on page 33](#) for information about using MSGLEVEL in a TP message log definition.
- The following JOB statement keywords have no effect in the APPC scheduling environment. APPC provides equivalent function elsewhere, except for the NOTIFY, RD, RESTART, and SECLABEL keywords.

### Keyword

#### Description

#### USER

Identifies job submitter's ID to system

#### GROUP

Identifies job submitter's GROUP to system

#### PASSWORD

Identifies job submitter's RACF password to system

#### SECLABEL

Security classification for this job

#### NOTIFY

TSO user ID notification when job completes (See note [“2” on page 70.](#))

#### CLASS

Assigns job to a job class

#### MSGCLASS

Assigns job log to an output class

#### RD

Options to control job checkpoints and restarting

#### TYPRUN

Requests special job processing

**PRTY**

Assigns JES selection priority

**RESTART**

Specifies job restart point

**Note:**

1. Even though these keywords have no effect, if found within the JCL, they must be syntactically correct to avoid a JCL error.
2. Coding the NOTIFY parameter might cause a runtime error, as well as a JCL syntax error; read [“Restrictions on &SYSUID Variable” on page 69](#).

**EXEC Statement Restrictions**

The EXEC statement RD keyword, which specifies options to control JOB checkpoints and restarting, has no effect in the APPC scheduling environment. However, if you code the RD keyword, it must be syntactically correct to avoid a JCL error.

**DD Statement Restrictions**

The following DD statement keywords have no effect or function inconsistently in the APPC scheduling environment:

**Keyword****Description**

**\***

Specifies in-stream data to follow

**DATA**

Specifies in-stream data to follow

**DLM**

Specifies the delimiter to terminate in-stream data

**TERM=TS**

Dataset represents Input/Output with a TSO terminal

**SUBSYS**

Symbolic substitution is inconsistent when you code &SYSUID as a subparameter.

**Note:**

1. Even though these keywords have no effect, if found within the JCL, they must be syntactically correct to avoid a JCL error.
2. Instream DD \* or DD DATA statements are processed as JCL statements and could cause JCL errors. The /\* statement (with a blank as the third character) is ignored.

**OUTPUT Statement Restrictions**

- The OUTPUT statement OUTDISP keyword has limited effect: In an APPC scheduling environment, SYSOUT data sets are treated as spin data sets. The system will process only the normal output disposition. If you code an abnormal output disposition, the system will check it for syntax and ignore it. If you code OUTDISP for either output disposition, it must be syntactically correct to avoid a JCL error.
- The following OUTPUT statement keywords have different default and override values in an APPC scheduling environment.

ADDRESS  
BUILDING  
DEPT  
NAME  
ROOM

See [z/OS MVS JCL Reference](#) for specific defaults and overrides.

## PROCLIB Restrictions

Although the TP profile can define resources with JCL, it cannot invoke a system PROCLIB. To specify a procedure in the JCL statements, explicitly include a JCLLIB statement to specify the dataset name or names where the JCL is to be found. An example of a JCLLIB statement follows:

```
//CHECKTP JOB (5523),MSGLEVEL=(1,1)  
//      JCLLIB ORDER=(FRED.PROCLIB,BILL.PROCLIB,JOE.PROCLIB)
```

**Note:** Using JCLLIB statements degrades performance. In addition, if you make changes to the procedures invoked by a multi-trans TP, the changes do not take effect until you modify the TP profile and the multi-trans TP stops and restarts.



---

## Chapter 6. Using the APPC/MVS Administration Utility

The APPC/MVS administration utility (ATBSDFMU) provides services in the form of commands that create and maintain TP profiles and side information through a batch job, an application program, or a REXX exec. Using the utility requires knowledge of JCL. An installation might choose to create the TP profiles and side information using the utility, and view and maintain them interactively with the dialog. For information about the administration dialog, see [Chapter 7, “Using the APPC/MVS Administration Dialog,” on page 83](#).

Topics covered in this chapter include:

---

### Utility Commands

The utility commands create and maintain TP profile and side information entries that are stored in VSAM key sequenced data sets. Each entry begins with a key, which identifies the entry within the file, and is followed by data specific to either the TP profile or side information. For information about the data within entries, see [“Creating a TP Profile” on page 59](#) and [“Creating Side Information” on page 65](#).

Utility commands also create and maintain the database tokens used to represent the VSAM file names in security definition statements.

The following are descriptions of the commands for TP profiles, side information, and database tokens.

### TP Profile Commands

#### **TPADD— TP Profile Addition**

Adds a TP profile to a VSAM file. During TP profile addition, the profile is validated to ensure that it is syntactically correct. If the profile is not syntactically correct or is already in the file (in either an activated or deactivated state), the TP profile is not added. When TP profiles are added on a lower level system than MVS/ESA SP 4.2, the utility cannot check the scheduler JCL portion for correct syntax.

If the TP is not scheduled by the APPC/MVS transaction scheduler, specify a scheduler exit with the TPSCHED\_EXIT keyword on the same line as TPADD.

#### **TPALIAS— TP Profile Alias Addition**

Adds an alias of an existing TP profile to the VSAM file. A TP profile alias allows a TP to be accessed by a key other than the key in its TP profile. The TP profile alias is not added when one of the following conditions exist:

- The alias to be added is already in the file
- The existing TP profile is not in the file
- The existing TP profile is an alias of another TP profile.

#### **TPDELETE— TP Profile Deletion**

Deletes a TP profile or a TP profile alias from a VSAM file. A user level TP that is registered for test, however, cannot be deleted.

#### **TPKEYS— TP Profile Keys Retrieval**

Retrieves the TP profile keys for all the TP profiles in a VSAM file for which a user has access authority. Retrieving TP profile keys provides you with a list of the profiles to which you have access. The profile keys are placed in a specified SYSSDOUT data set.

#### **TPMODIFY— TP Profile Modification**

Modifies a TP profile that already exists. You can specify one or more keyword changes for modification. Keywords not specified are not modified. During TP profile modification, the profile is validated to ensure that it is syntactically correct. If the profile is not syntactically correct or is not

found in the VSAM file, the TP profile is not modified. When TP profiles are modified on a lower level system than MVS/ESA SP 4.2, the utility cannot check the scheduler JCL portion for correct syntax.

#### **TPRETRIEVE— TP Profile Retrieval**

Retrieves a TP profile from a VSAM file and places it in a specified SYSSDOUT data set. Retrieving a TP profile allows you to view the contents of the TP profile.

## **Side Information Commands**

#### **SIADD— Side Information Addition**

Adds side information to a VSAM file. If the side information to be added is already in the file, the side information is not added.

#### **SIDELETE— Side Information Deletion**

Deletes side information from a VSAM file.

#### **SIKEYS— Side Information Keys Retrieval**

Retrieves from a VSAM file the side information keys for all of the side information for which a user has access authority. Retrieving side information keys provides you with a list of the side information to which you have access. The side information keys are placed in a specified SYSSDOUT data set.

#### **SIMODIFY— Side Information Modification**

Modifies side information that already exists. You can specify one or more keyword changes for modification. Keywords not specified are not modified.

#### **SIRETRIEVE— Side Information Retrieval**

Retrieves side information from a VSAM file and places it in a specified SYSSDOUT data set. Retrieving side information allows you to view the contents of the side information.

## **Database Token Commands**

The use of database tokens allows an installation to assign different levels of security to TPs within a single VSAM file. Each VSAM file can be assigned one database token. The database token represents the file name in security definitions.

#### **DBRETRIEVE— Database Token Retrieval**

Retrieves the database token for a VSAM file and places it in a specified SYSSDOUT data set. Retrieving a database token allows you to view the database token for the VSAM file.

#### **DBMODIFY— Data Base Token Modification**

Modifies the database token for a VSAM file. If the file does not already contain a database token, the token is added. The keyword for this command is DBTOKEN.

**Note:** Be careful when changing the database token for an existing VSAM file. Authorization to read and write to entries in VSAM files is associated with the database token through RACF APPCSI and APPCTP classes or equivalent classes in other security products. To maintain continued access after changing a database token, you must ensure that entities within the APPCSI and APPCTP classes reflect the new database token name to provide equivalent protection. To make this change with RACF, issue an RDEFINE command with the FROM keyword, followed by an RDELETE command.

## **Syntax Requirements**

---

To use the APPC/MVS administration utility to create and maintain TP profiles and side information, combine the appropriate utility command with the required TP profile and side information.

Utility commands can be issued in the JCL SYSIN data stream of the batch job that invokes the APPC/MVS administration utility. Following the command are the TP profile and side information keys and keywords required for the command. Not all commands require keys or keywords.

```
//STEP    EXEC PGM=ATBSDFMU
...

//SYSIN   DD   *
Utility command
            Key(parameter)
```



```

Keyword(keyword_parameter)
Keyword(keyword_parameter)
Keyword(keyword_parameter)
/*

```

Information on the same line as the utility command name is ignored except for the TPSCHEXIT keyword for TPADD and TPMODIFY.

Specific commands require specific data. The order of the required data is not significant except for TPALIAS, in which case the alias TP profile key must appear before the existing TP profile key. [Table 6](#) on [page 75](#) maps the required data to the command; use the figure along with the examples under “Examples” on [page 80](#) to understand the syntax for specific utility commands.

Table 6. APPC/MVS Administration Utility Commands and Required Data	
Command	Required Data
TPADD	TP profile key, TP profile keywords for TPs scheduled by the APPC/MVS transaction scheduler:  TPSCHEX_DELIMITER CLASS (if no parmlib default) JCL_DELIMITER , Scheduler JCL JOB and EXEC statements
TPALIAS	TP profile key (alias), TP profile key (existing)
TPDELETE	TP profile key
TPKEYS	None
TPMODIFY	TP profile key, TP profile keywords to be modified
TPRETRIEVE	TP profile key
SIADD	Side information key, side information (all keywords)
SIDELETE	Side information key
SIKEYS	None
SIMODIFY	Side information key, side information keywords to be modified
SIRETRIEVE	Side information key
DBMODIFY	Database token
DBRETRIEVE	None

## Syntax Rules

The following general rules apply to utility command syntax.

- Blank lines can appear anywhere within the SYSIN data.
- Only one command or keyword is allowed per line.
- No continuation to the next line is allowed for command lines, keyword lines, and comment lines.
- The system does not recognize command syntax entered in columns 73 through 80, except for JCL records within the JCL\_DELIMITER area/keyword.
- Data can start in any column. The exceptions to this rule are the data delimiters specified by the TPSCHEX\_DELIMITER keyword and the JCL\_DELIMITER keyword. The specified delimiter must start in the first column.
- Lowercase characters are accepted as equivalent to uppercase when used in command names, keywords, and most keyword values. An exception to this is that lowercase and uppercase characters are distinguished in the TPNAME, the TPSCHEX\_DELIMITER, and the JCL\_DELIMITER keyword values.

- Syntax errors are written as messages to SYSPRINT. After the first error message, checking of syntax continues until all input is checked.

### For Comments

- Comment lines are preceded by /\*
- Comments are not allowed on the same line as keywords or utility commands. A comment must appear on its own line.
- Comment lines and blank lines may appear on any line of the SYSIN data, but the comment delimiter /\* must not begin in column 1.

### For Commands

- The first line (not including blank or comment lines) of utility input must name the utility command.
- Multiple commands are allowed within a single SYSIN data stream.
- A utility command must be followed by all of the keyword lines that are required for the command before the next command is specified or before the end of input.

### For Keywords

- If a keyword value is not valid, or a keyword is missing, the keyword line is not valid, and the entire command is not valid.
- Only keywords required or optional for a given command can be specified.
- A keyword cannot be repeated in the context of a single command. (Exception: The TPNAME keyword does appear twice for the TPALIAS command.)
- Do not code extraneous data (any data that follows the last right parenthesis) on a keyword line. Extraneous data is ignored but causes a warning message.
- Zero or more blanks are allowed between the keyword and the left parenthesis in a keyword line.
- Keywords may appear in any order following a command. There are two exceptions to this rule:
  - When the TPALIAS command is specified, all keywords for the alias TP profile key must appear before any keywords for the existing TP profile key.
  - When a transaction scheduler section is present, all of its associated keywords followed by its corresponding data delimiter must appear separately from any keywords associated with the TP profile key or attributes section. That is, keywords from the TP profile key or attributes section cannot appear within the transaction scheduler section, and vice versa.

## Invoking the APPC/MVS Administration Utility

---

You can invoke the APPC/MVS administration utility through:

- A batch job
- An application program
- A REXX exec

This section explains each method, the restrictions for invoking the utility, and the return codes you receive after utility processing completes.

If you are using the program access to data sets (PADS) function of RACF to protect access to TP profiles and side information, see [“Giving Program Access to the APPC/MVS Administration Utility” on page 149](#) for more information about using the administration utility in that environment.

## Invoking the APPC/MVS Administration Utility from a Batch Job

The APPC/MVS administration utility is invoked from a batch job by naming ATBSDFMU on the EXEC statement. The utility commands are issued either in the SYSIN data stream or from a data set named in the SYSIN DD statement.

```
//jobname JOB ...  
//stepname EXEC PGM=ATBSDFMU,PARM='TYPRUN=condition'  
//SYSSDLIB DD DSN=SYS1.APPCTP,DISP=SHR  
//SYSSDOUT DD DSN=UTILITY.OUTPUT,DISP=(NEW,CATLG)  
//SYSPRINT DD SYSOUT=A  
//SYSIN DD DATA,DLM=XX
```

*Utility command and  
TP profile data or  
side information data*

**XX**

Figure 56. Example of JCL to Invoke ATBSDFMU Using SYSIN Data

```
//jobname JOB ...  
//stepname EXEC PGM=ATBSDFMU,PARM='TYPRUN=condition'  
//SYSSDLIB DD DSN=SYS1.APPCTP,DISP=SHR  
//SYSSDOUT DD DSN=UTILITY.OUTPUT,DISP=(NEW,CATLG)  
//SYSPRINT DD SYSOUT=A  
//SYSIN DD DSN=TP.INSTALL,DISP=SHR
```

Figure 57. Example of JCL to Invoke ATBSDFMU Using a SYSIN Data Set

Before invoking the utility, read [“Restrictions on Invoking the APPC/MVS Administration Utility” on page 80.](#)

### Parameters

#### jobname

The name of the job that invokes the APPC/MVS administration utility.

#### stepname

The name of the step that invokes the APPC/MVS administration utility (ATBSDFMU) and specifies a TYPRUN condition. TYPRUN controls the action the utility performs on the profile and side information data entered as SYSIN input.

#### condition

Indicates one of three TYPRUN actions (on the EXEC statement) for the SYSIN input:

##### SCAN

Syntax check only the utility command and keywords.

##### APPC

If MVS/ESA SP 4.2 or later has been installed, perform all services. If MVS/ESA SP 4.2 or later has not been installed, perform all services except TP profile addition and modification services. TP profile addition and modification services are scanned only.

##### RUN

Perform all services regardless of the release level. However, if MVS/ESA SP 4.2 or later has not been installed, the JCL portion of the transaction scheduler section in TP profiles cannot be validated. If an attempt is made to run the TP with JCL syntax errors, the TP fails with no JCL messages to indicate the nature of the problem.

RUN is the default for TYPRUN and the recommended parameter when using the utility on a lower level system.

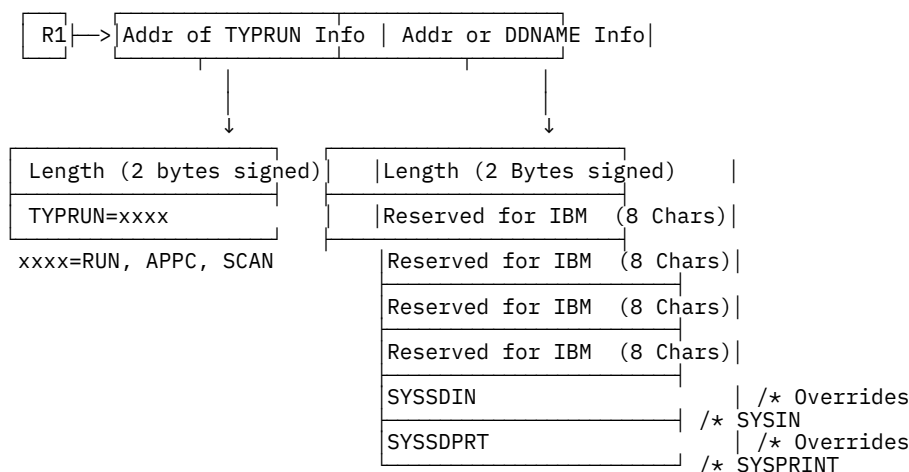
## SYSSDLIB

Identifies the VSAM KSDS into which definitions will be placed or from which definitions are retrieved. The VSAM KSDS specified must have been previously created.

## SYSSDOUT

Identifies a data set that will receive output from the APPC/MVS administration retrieval services. The data set can be previously allocated with a recommended format of fixed blocked records with a record length of at least 120 bytes.

Retrieved output consists of /\* and the command name that requested the output, the SYSIN input for the service, and a blank line followed by the requested output. For example, output for a TPRETREIVE is displayed as follows:



## SYSPRINT

Identifies a data set that can receive general statement and message output from the APPC/MVS administration utility. The data set can be previously allocated with a recommended format of fixed blocked records with a record length of at least 120 bytes.

A SYSPRINT DD statement is required for all administration utility services. The default message output class for SYSPRINT is SYSOUT=A.

## SYSIN

Identifies the input stream or name of a data set in which the APPC/MVS administration utility command is specified along with the necessary TP profile or side information data. Some commands, such as the key retrieval commands, require no TP profile or side information data. Most of the commands, however, require a part or all of the TP definition.

**Note:** In case of a severe error returned by the APPC/MVS administration utility, you can attempt to capture a dump by including one of the following DD statements:

```
//SYSABEND DD ...
//SYSMDUMP DD ...
//SYSUDUMP DD ...
```

For more information, see [z/OS MVS JCL Reference](#).

## Invoking the APPC/MVS Administration Utility from an Application Program

In addition to invoking the administration utility (ATBDSDFMU) from a batch job, you can invoke the utility from within an application program by using the LINK macro.

Before invoking ATBDSDFMU, you must allocate the following files:

- SYSSDIN

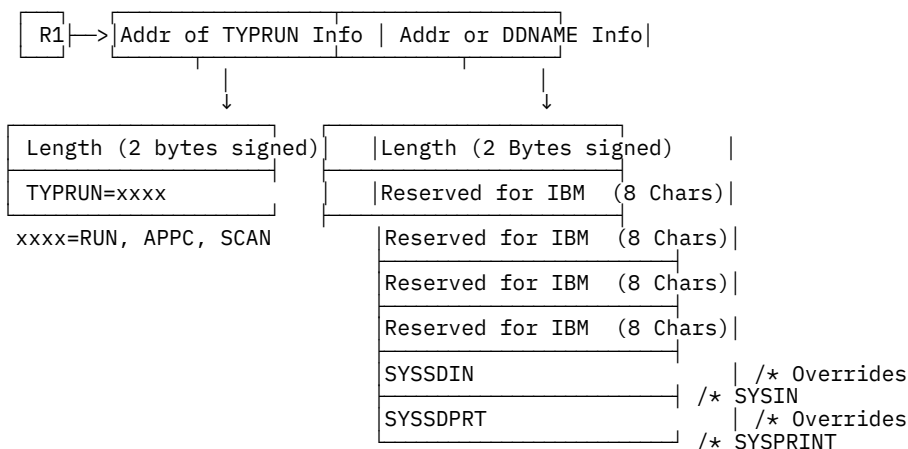
- SYSSDLIB
- SYSSDPRT
- SYSSDOUT

You can allocate these files dynamically from within your program, or you can submit the appropriate JCL as in the following example:

```
//jobname JOB ...
//stepname EXEC PGM=MYPROG
//SYSSDLIB DD DSN=SYS1.APPCTP,DISP=SHR
//SYSSDOUT DD DSN=MYPROG.OUTPUT,DISP=(NEW,CATLG)
//SYSSDPRT DD DSN=MYPROG.SYSPRINT,DISP=(NEW,CATLG)
//SYSSDIN DD DSN=MYPROG.SYSIN,DISP=SHR
```

Note that before invoking ATBSDFMU, the caller must either be in supervisor state, or run with PSW key 0 or 1.

The caller must also build, in key 1 storage, a parameter list that the caller passes to ATBSDFMU. This parameter list must contain the following:



Because the SYSIN and SYSPRINT parameters are already being used by the system, you must specify alternate variable names in your parameter list to avoid any conflicts. In this parameter list, SYSSDIN and SYSSDPRT override SYSIN and SYSPRINT.

For a complete description of the TYPRUN, SYSIN, and SYSPRINT parameters, see [“Parameters” on page 77](#).

## Invoking the APPC/MVS Administration Utility from a REXX Program

Instead of invoking the APPC administration utility directly, you can use the REXX language to invoke ICQASLI0, the interface to the utility. ICQASLI0 requires you to allocate the following four files:

- SYSSDIN
- SYSSDLIB
- SYSSDPRT
- SYSSDOUT

For more information about these files, see [“Parameters” on page 77](#).

The following example shows a portion of a REXX exec that invokes the APPC Administration Utility. Because ICQASLI0 sets TYPRUN=RUN (default), you do not need to specify this parameter.

```
ADDRESS TSO

"Alloc da(TEMP.SYSSDIN) file(SYSSDIN) shr reuse"
"Alloc da(TEMP.SYSSDOUT) file(SYSSDOUT) shr reuse "
"Alloc da(TEMP.SYSSDPRT) file(SYSSDPRT) shr reuse "
"Alloc da(TEMP.APPCTP) file(SYSSDLIB) shr reuse"
```

```
"icqasli0"

if rc ^= 0 then
  say "Utility Operation failed."
else
  say "Utility Operation succeeded."

'Free f(syssdin syssdlib syssdpri syssdout)'
```

## Restrictions on Invoking the APPC/MVS Administration Utility

During add, delete, modify, or alias command processing, the APPC/MVS administration utility updates the VSAM KSDS, but it updates in-storage profiles within only the APPC address space that runs on the same system as the utility. If other systems share a VSAM KSDS for TP profile or side information, those systems cannot detect the updates.

When TP profile or side information data sets are shared among systems, IBM strongly recommends that you do not use the utility to update them. Damage might result if you use the utility to update a data set while systems are sharing it. (However, you can safely use the retrieve and key utility commands whether the data set is shared or is used by only one system.)

To safely update a data set that is shared among systems:

1. Copy the shared data set into another VSAM KSDS
2. Use the utility to update the copy
3. Make the copy available for use by doing the following **on each system** that shares the data set:
  - a. Create an APPCPMxx parmlib member that names the updated data set
  - b. Issue a SET command to activate the new parmlib member.

## Return Codes

APPC/MVS administration utility commands may return the following return codes in register 15 or display them in the job output. Hexadecimal values are in parentheses in the following list.

### Return Code Meaning

- 0 (0)**  
Processing successful.
- 4 (4)**  
Processing successful. Warning message or messages issued.
- 8 (8)**  
Processing unsuccessful. At least one request failed.
- 12 (C)**  
Processing unsuccessful. Severe error; processing terminated.

## Examples

SYS1.SAMPLIB member ATBUTIL contains several examples that use the APPC/MVS administration utility to create, maintain, and delete TP profiles and side information. The examples show how to use utility commands to:

- Add a TP profile for a standard transaction program.
- Add a TP profile for a system-level multi-trans transaction program.
- Add two TP profiles for a non-APPC/MVS transaction scheduler.

In this example, the transaction scheduler name is XYZ, and the XYZ profile has three defined inputs: TRANSACTION\_CODE\_NAME, SCHEDULE\_CLASS, and MAX\_REGIONS. The TPSCHED\_EXIT keyword specifies a parse routine, XYZEX01, to check input that is specific for the scheduler (if the keyword is omitted, the APPC/MVS transaction scheduler is assumed).

- Add a TP profile as an alias of an existing TP profile.
- Modify a TP profile by specifying the TP key and the keyword to be modified.
- Modify side information by specifying the side information key and the keyword to be modified.
- Add side information to a VSAM file.
- Retrieve a TP profile from a VSAM file.

With the following changes, this example can also show how to retrieve side information:

- Change the TPRETRIEVE command to SIRETRIEVE
- Change the SYSSDLIB data set name to one for side information
- Replace the TP profile key with a side information key.
- Delete side information from a VSAM file.

With the following changes, this example can also show how to delete a TP profile:

- Change the SIDELETE command to TPDELETE
- Change the SYSSDLIB data set name to one for TP profiles
- Replace the side information key with a TP profile key.
- Retrieve TP profile keys from a VSAM file.

With the following changes, this example can also show how to retrieve side information keys:

- Change the TPKEYS command to SIKEYS
- Change the SYSSDLIB data set name to one for side information.
- Retrieve the database token from a VSAM file.

For more information about setting the security environment related to RACF profiles, see [Chapter 10, “Setting up Network Security,”](#) on page 133.

- Modify the database token in a VSAM file.

Be careful when changing the database token for an existing VSAM file. See the note at [DBMODIFY–Data Base Token Modification](#) for details.





# Chapter 7. Using the APPC/MVS Administration Dialog

The APPC/MVS administration dialog, from here on called the dialog, is a panel interface to the APPC/MVS administration utility.

When you use the utility, you specify a utility command and keywords through the SYSIN DD statement of a batch job; with the dialog, you can select comparable services from panels. The dialog does not require extensive knowledge of JCL, and it can be used interactively in a TSO/E session.

Topics in this chapter include:

References:

[z/OS ISPF User's Guide Vol I](#)  
[z/OS TSO/E Administration](#)  
[z/OS TSO/E Customization](#)

## Overview of the Dialog

This chapter provides a brief introduction to the dialog and uses a subset of the panels from the dialog to illustrate their use. Before using the dialog, read “Restrictions on Invoking the APPC/MVS Administration Utility” on page 80, which also applies to the dialog.

From the main selection panel of the dialog, you can choose one of three forms of administration: TP profile, side information, or database token.

```
ICQASE02                                APPC Administration

Command ==>

Select one of the following with an "S".  Then Enter.
Type information.  Then Enter.

- TP Profile Administration
  Current TP Profile
  System file . .  SYS1.APPCTP_____

- Side Information Administration
  Current Side Information
  System file . .  SYS1.APPCSI_____

- Database Token Administration
  Current Database Token
  System file . .  SYS1.APPCTP_____

Note: For a list of file names, add an "*" suffix to the partial data set name.

PF1=Help  PF3= Exit  PF12= Cancel
```

The main selection panel leads to the following dialog services; some have comparable utility command names, but others provide services not available with the utility.

### TP Profile Administration

Table 7. TP Profile Administration	
Dialog Service	Utility Command
List TP names and levels from a TP profile file	TPKEYS
Add a new TP profile	TPADD

<i>Table 7. TP Profile Administration (continued)</i>	
<b>Dialog Service</b>	<b>Utility Command</b>
Copy an existing TP profile to create another entry	None
Edit an existing TP profile	TPMODIFY
Browse a TP profile	TPRETRIEVE
Delete a TP profile	TPDELETE
Add a TP profile alias	TPALIAS

## Side Information Administration

<i>Table 8. Side Information Administration</i>	
<b>Dialog Service</b>	<b>Utility Command</b>
List symbolic destination names from a side information file	SIKEYS
Add new side information	SIADD
Copy existing side information to create another entry	None
Edit existing side information	SIMODIFY
Browse side information	SIRETRIEVE
Delete side information	SIDELETE

## Database Token Administration

<i>Table 9. Database Token Administration</i>	
<b>Dialog Service</b>	<b>Utility Command</b>
Display a database token	DBRETRIEVE
Modify a database token	DBMODIFY

## How to Use the Dialog

The dialog is a set of predefined display images, called panels, from which you can add and delete information. Each panel contains directions for using the functions on the panel. If you are unsure about what to do on the panel, you can obtain help information for each panel by typing **help** on the command line or by pressing the HELP PF key.

How you access the dialog depends on where it was installed. For information about installing the dialog and how to access it, see [“Installing the Dialog”](#) on page 87.

Most of the dialog's panels accept user input through a command line, PF keys, and input fields. In addition, messages appear when an error occurs and when a function completes.

## Using a Command Line

The command line can appear at either the top or the bottom of the screen, depending on the user's ISPF screen display characteristics. From the command line, you can issue any ISPF command, a TSO/E command that is prefaced by “tso”, and the REQAPPC command.

Use the REQAPPC command when the dialog is installed with a level of MVS that is lower than MVS/ESA SP 4.2. REQAPPC ON and REQAPPC OFF control whether you require APPC to syntax check the JCL portion of a TP profile. When a TP profile is added or modified and MVS/ESA SP 4.2 or later has not been

installed, the scheduler JCL cannot be syntax checked. To override the requirement for syntax-checked JCL, you can type REQAPPC OFF and the TP profile is added or modified with no syntax checking. The default is REQAPPC ON.

## Using PF Keys

Program function (PF) keys are keyboard equivalents to commands. PF keys and the ENTER key allow a user to move from panel to panel, and to perform functions more quickly than if they were typed on the command line.

PF key default values for the APPC administration dialog are set in the non-display panel (ICQASE00). Descriptions of ENTER key processing and default processing for the PF keys follow:

### ENTER key

Processes information typed on the panel and displays the next panel.

### Help PF key

Displays a help panel for a functional panel or currently displayed message.

### Exit PF key

Cancels a function (possibly several panels' worth of input) and returns the user to the most recently viewed primary panel from which the function was selected. Exit is equivalent to End.

### Cancel PF key

Cancels processing on the currently displayed panel and returns the user to the previously displayed panel.

## Using Input Fields

Upon initial display, input fields on the panels contain defaults where applicable. If you type over the default, the input field retains the new value until the administrative function is completed or cancelled, after which the default value is re-displayed. For example, defaults on the first panel for adding a TP profile are initially displayed as follows:

```
ICQASE08                                ADD TP Profile

Command ==>                                More ...

Type information.  Then Enter.

Transaction Scheduler . . ASCH_____
To system file . . SYS1.APPCTP_____
TP Name _____
Level . . . . SYSTEM      System/Group/User
ID . . . . . _____    Group ID or User ID
                               (required for Group or User Level)

Active Status . . YES      Yes/No

PF1=Help  PF3= Exit  PF12= Cancel
```

When the default value in the Active Status field is written over with a **NO**, that NO will remain for the duration of adding the TP profile. After the TP profile is added or if the addition is cancelled from one of the panels, the Active Status field reverts back to the default **YES**.

The panel provides information about what each input field can contain. More information is available from the help panel. For example, one of the help panels for the Add TP Profile panel provides the following information:

ICQBS081  
HELP  
Command ==>

APPC Administration

Page 2 of 3

Add TP Profile  
(continued)

Enter the following fields:

		Notes
TP Name	A-Z a-z 0-9 ( . < + & * ) ; - / , % _ > ? ' = " :	An asterisk (*) is not a recommended value in this field.
Level	SYSTEM GROUP USER	
ID	A-Z 0-9	Required for GROUP or USER level. For GROUP, the first character of ID must be A-Z.
Active Status	YES NO Y N	

When Active Status is YES or Y, the TP can be accessed and run. If the TP needs to be inactivated because of problems within the TP or in the resources it uses, you can change the Active Status to NO or N.

Press ENTER for more HELP.

Press EXIT to leave HELP.

## Receiving Messages and Getting Help

If an input field contains incorrect input, or if a function does not process correctly, a message appears on the panel.

ICQASE08

ADD TP Profile

**TP Profile already exists. ICQAS506**

Command ==>

More ...

Type information. Then Enter.

Transaction Scheduler . . ASCH\_\_\_\_\_

To system file . . SYS1.APPCTP\_\_\_\_\_

TP Name **newTP**\_\_\_\_\_

Level . . . . SYSTEM System/Group/User

ID . . . . . \_\_\_\_\_ Group ID or User ID  
(required for Group or User Level)

Active Status . . YES Yes/No

PF1=Help PF3= Exit PF12= Cancel

To obtain more information about the message, press PF1 to display a help panel.

ICQ506AS  
HELP  
Command ===>

Message = ICQAS506

Page 1 of 1

**TP Profile already exists. ICQAS506**

Explanation: The attempt to add a TP profile failed because the TP profile already exists for the specified combination of "To" system data file and TP profile identification (TP Name, Level, ID).

Browse the existing TP profile entry and, if necessary, edit the existing entry or add a new entry under a unique combination of TP Name, Level, ID and "To" system data file.

Press ENTER for more HELP.  
Press EXIT to leave HELP.

## Installing the Dialog

The dialog can be installed as an application under Application Manager in the Information Center Facility, or as an option from an ISPF menu. In either case, TSO/E 2.3 or later and the Information Center Facility must be installed before you install the dialog.

When the dialog is installed under Application Manager:

- The dialog can be restricted to administrators who can access the administrator's menu, and the dialog can be further restricted within the Application Manager.
- An installation file (ICQFF007) in ICQ.ICQILIB is provided for easy installation under Application Manager.

For information about installation files, see [z/OS TSO/E Administration](#).

The APPC/MVS administration dialog is shipped with the Information Center Facility. An installation file is shipped with TSO/E and is provided for installing the APPC/MVS administration dialog on the Information Center Facility as an option on the main administrator panel, ICQADMIN. For a description of this installation process, see [z/OS TSO/E Customization](#).

When the dialog is installed as an option from an ISPF menu:

- The dialog can be made accessible to anyone who uses ISPF.
- No installation file is provided.

The APPC/MVS administration dialog can also be invoked as a REXX exec from an ISPF panel. The customization requirements (TSO/E and ISPF) for invoking the APPC/MVS administration dialog directly from an ISPF panel are illustrated in the examples that follow.

## Installing the Dialog under Application Manager

The TSO/E Information Center Facility program ICQASLI0 which is used by the dialog must be added to the list of names of programs that must be authorized when called by the TSO/E Service Facility. The same IKJTSOxx member that follows shows this specification requirement.

```

/*****
/*MVSAPPC
/* 1. Authorize the command list
/* 2. Authorize the program list
/* 3. Identify the list of commands that cannot be issued in the
/*    background.
/* 4. Create the defaults for SEND command
*****/

AUTHCMD NAMES(          /* AUTHORIZED COMMANDS */      +
  RECEIVE              /*    TSO COMMANDS    */      +
  TRANSMIT XMIT        /*                      */      +
  .
  .
  .

AUTHPGM NAMES(          /* AUTHORIZED PROGRAMS */      +
  GIMSMP               /* S.M.P./E           */      +
  .
  .
  .

NOTBKGND NAMES(        /* COMMANDS WHICH MAY NOT BE */      +
                      /* ISSUED IN THE BACKGROUND */      +
  OPER      OPERATOR    /*                      */      +
  TERM      TERMINAL)   /*                      */      +

AUTHTSF NAMES(         /* PROGRAMS TO BE AUTHORIZED WHEN */      +
                      /* CALLED THROUGH TSO SERVICE FACILITY*/      +
  ICQASLI0          /* FOR APPC/MVS DIALOG */      +
  .
  .
  .

SEND              /* SEND COMMAND DEFAULTS */      +
  OPERSEND(ON)    /*                      */      +
  .
  .
  .

```

Figure 58. Sample IKJTSOxx PARMLIB member

Optionally, the default values used for allocation and selection fields in the dialog may be changed by updating member ICQASE00 in the Information Center Facility panel library. The sample member that follows illustrates some of the defaults that may be changed. For a complete description of these specifications, see [z/OS TSO/E Customization](#).

```

/* NAME:  ICQASE00

)BODY
)INIT
/*****
/*The following customizable values are used in the allocation      */
/*of the datasets for APPC Administration:                          */
/*                                                                    */
/* SYSSDIN  - Input to the APPC Administration Utility              */
/* SYSSDOUT - Output from the APPC Administration Utility          */
/* SYSSDPRT - Output from the APPC Administration Utility          */
/* SYSSDATA - JCL for ASCH TP Profile or Non-ASCH scheduler data   */
/*                                                                    */
/*Recommendations: Do not decrease space to less than 50,10      */
/*****
&QASSDIN  = 'BLKSIZE(3120) SPACE(50,10)'
&QASSDOUT = 'BLKSIZE(3120) SPACE(50,10)'
&QASSDPRT = 'BLKSIZE(3120) SPACE(50,10)'
&QASSDATA = 'BLKSIZE(3120) SPACE(50,10)'

VPUT (QASSDIN QASSDOUT QASSDPRT QASSDATA) SHARED
:
/*****
/* QASCLASS is a customizable list containing values for the      */
/* scheduler class field of an "ASCH" TP Profile.                  */
/* These values represent the scheduler class when the TP is       */
/* attached.                                                         */
/*****
&QASCLASS = 'A B'

VPUT (QASCLASS) SHARED

/*****
/* Default dataset name for Model JCL                               */
/*****
&QASMODDF = 'APPC.TPMODEL.JCL'

VPUT (QASMODDF) SHARED

/*****
/*Default dataset names for the system data files.  If the system */
/*file is blank, these names will be substituted.                  */
/*****
&QASTPDEF = 'APPC.APPCTP'
&QASSIDEF = 'APPC.APPCSI'
&QASDBDEF = 'APPC.APPCTP'

VPUT (QASTPDEF QASSIDEF QASDBDEF) SHARED
:
)PROC
)END

```

Figure 59. Sample ICQASE00 ICF Member

## Installing the Dialog from ISPF

There are two methods of installing the dialog from ISPF: use the TSO/E ICF or invoke it from an ISPF selection panel.

- Enter **TSO ICQASRM0** from any ISPF command line, or
- Invoke the dialog from an ISPF selection panel.

ISPF 2.3 or later is required to provide support for the dialog. Following are a sample ISPF selection panel definition and an actual panel that was used to invoke the dialog from ISPF.

```

)ATTR DEFAULT(%_)
# type(text) intens(high) color(red)
$ type(text) intens(high) color(yellow)
@ type(text) intens(low) color(green)
? type(text) intens(non) color(pink)
\ type(text) intens(non) color(blue)
)BODY
%-----Software House, Inc.  APPC/MVS Development Options----
%OPTION  ===>_ZCMD

#APPC      - #APPC/MVS Administration Dialog      #D      - $SDSF
#APPCAST   - #APPC Assist Dialog                  #IPCS   - #I.P.C.S.
#APPCTELL  - #APPCTELL TP
$
)INIT
.HELP=WSCHMODS
)PROC
&ZSEL=TRANS( TRUNC (&ZCMD, '.')
              /*APPC/MVS Options Follow ****/
              APPC, 'CMD(%ICQASRM0)'              /* APPC/MVS */
              APPCAST, 'Panel(APPCAST)'           /* APPCAST */
              APPCTELL, 'CMD(%APPCTELL)'          /* APPCTELL */
              D, 'PANEL(ZSDSFOP2) NEWAPPL(ISF)'    /* S.D.S.F. */
              IPCS, 'PGM(BLSG) PARM(PANEL(BLSPRIM) NEWAPPL(BLSG)'
              X, 'EXIT'
              *, '?'
IF (&ZCMD = 'D')
&ZSEL = 'PGM(ISFISP) NOCHECK NEWAPPL(ISF)'
&ZTRAIL = .TRAIL
IF (&ZCMS = ' ')
IF (.RESP = ENTER)
&ZPARENT = ISR@PRIM
)END

```

Figure 60. Sample ISPF Panel Definition for APPC/MVS Selections

```

-----Software House, Inc.  APPC/MVS Development Options-----
OPTION  ===>

APPC      - APPC/MVS Administration Dialog      D      - SDSF
APPCAST   - APPC Assist Dialog                  IPCS   - I.P.C.S.
APPCTELL  - APPCTELL TP

```

Figure 61. Sample ISPF Selection Panel with APPC/MVS Options

## Customizing the Dialog

An installation can customize the dialog by changing the following variables contained in the non-display panel ICQASE00 in ICQ.ICQLIB. Variables you can change and their default values appear in [Table 10 on page 90](#).

Table 10. Variables in ICQASE00		
Variable	Contents	Default Value
QASSDIN	Allocation attributes for the data set used as input to APPC administration	BLKSIZE(3120) SPACE(50,10)
QASSDOUT	Allocation attributes for the data set used for SYSOUT output from APPC administration	BLKSIZE(3120) SPACE(50,10)
QASSDPRT	Allocation attributes for the data set used for SYSPRINT output from APPC administration	BLKSIZE(3120) SPACE(50,10)
QASSDATA	Allocation attributes for the data set used to contain the JCL for an ASCH TP profile or to contain the scheduler information for a non-ASCH TP profile	BLKSIZE(3120) SPACE(50,10)



Table 10. Variables in ICQASE00 (continued)		
Variable	Contents	Default Value
QASTSPE	<p>A map of transaction scheduler exits. The value on the left is the value used in the transaction scheduler field name. The value on the right is the program/module name of the transaction scheduler exit that is called by APPC administration to check the syntax of the scheduler data. For example, if the scheduler name is XYZ and the exit to check the syntax is XYZEX01, specify "XYZ,XYZEX01".</p> <p>You can have more than one transaction scheduler, but ASCH,ASCH is required. To specify more than one transaction scheduler, separate them with a + as in the following example:</p> <pre>' ASCH,ASCH +   ABC,ABCEXIT +   XYZ,XYZEX01 '</pre>	ASCH,ASCH
QASCLASS	The available classes for a TP scheduled by the APPC/MVS transaction scheduler. These classes must have been defined in one or more ASCHPMxx parmlib members. To allow the user to display a list of available classes, write the class names as values after this variable. To keep the list up-to-date, add or delete class names here when classes are added or deleted.	none
QASMODDF	The data set name that contains JCL models. As an aid to the person adding TP profiles scheduled by ASCH, sample JCL models can be created and placed in the data set named here.	ICQ.**
QASTPDEF	The name of the default VSAM KSDS for TP profile administration.	SYS1.APPCTP
QASSIDEF	The name of the default VSAM KSDS for side information administration.	SYS1.APPCSI
QASDBDEF	The name of the default VSAM KSDS for database token administration.	SYS1.APPCTP



---

## Part 3. Session management



---

## Chapter 8. Planning Sessions

To enable and support the flow of conversations between like and unlike systems, installations must define LUs between which sessions can bind. Planning the LUs and the sessions for the MVS portion of an APPC network involves certain decisions, such as how many local LUs to define on MVS. Fewer LUs are easier to administer, but many LUs allow for flexibility.

Sessions determine where conversations can flow and the way they flow. An installation can use defaults to determine session characteristics or it can customize session characteristics.

References:

- [\*z/OS Communications Server: SNA Customization\*](#)
- [\*z/OS Communications Server: SNA Network Implementation Guide\*](#)
- [\*z/OS Communications Server: SNA Resource Definition Reference\*](#)

---

### Determining the Number of Local LUs

The maximum number of local LUs you can define on a z/OS system is 500, of which up to 200 can be associated with a transaction scheduler. For administration simplicity, you should define the smallest number of LUs that you need for your installation. You must define at least one LU before APPC/MVS processing can take place, even when APPC processing remains on a single system.

Although APPC/MVS requires only one LU to start processing, an installation might need more than one LU for various reasons:

- **Testing applications**— Programmers who use a test system might want a separate LU from which to test applications. The test applications' profiles can be kept in a separate TP profile file as well. After testing the applications, programmers can add the applications' profiles to the production TP profile file.
- **More than one transaction scheduler**— Only one transaction scheduler can be associated with an LU at a time. If your installation uses a transaction scheduler in addition to the APPC/MVS transaction scheduler, define a separate LU for each scheduler.
- **Isolation of TPs accessed**— An LU can be created to provide exclusive access to a group of TPs. TPs are accessed from a single TP profile file named to an LU. To completely isolate a group of TPs from every other LU, place the TP profiles for the TPs in a separate file from other TP profiles, and associate that TP profile file with one specific LU.

To further isolate TPs within the file, coordinate the TP level from its TP profile key with the LU level. For information about levels, see [“TP Profile Key”](#) on page 59.

- **Using APPC/MVS Servers**— Servers cannot normally receive inbound requests from LUs that are associated with a transaction scheduler (unless the servers are associated with the scheduler). If your installation uses one or more APPC/MVS servers, define at least one LU that is not associated with a transaction scheduler. Such LUs are called NOSCHED LUs.
- **Outbound Requests When No Scheduler is Active**— To allow TPs to flow outbound allocate requests when no transaction scheduler is active, define one NOSCHED LU.

---

### Defining the System Base LU

The system base LU is the default LU for handling outbound work not already associated with a particular LU. The LUADD statements in the active APPCPMxx parmlib member (or members) determine which LU is the system base LU:

- The system base LU is represented by the last LUADD statement that contains both the NOSCHED and BASE parameters. This type of system base LU allows outbound requests to be processed when no transaction schedulers are active.

- If no LUADD statements contain both NOSCHED and BASE, the system base LU is represented by the last LUADD statement that contains the BASE parameter and specifies— either explicitly or by default—the APPC/MVS transaction scheduler (ASCH).

If neither of these types of system base LU are defined in the active configuration, APPC/MVS rejects Allocate requests for outbound conversations from MVS programs (TSO/E users, started tasks, and other work requests) that are not associated with a scheduler or an LU. The Allocate callable service description, in *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*, explains which local LUs can be specified or used by default for outbound conversations.

MVS TPs that are scheduled in response to incoming allocate requests do not use a base LU. When an MVS TP receives an incoming allocate request, the request was sent over a session bound between two LUs, one of which was an MVS LU. When the MVS TP responds to the allocate request, it is automatically associated with the same MVS LU through which the request entered the system. Therefore, the only conversations that need a base LU are outbound conversations from programs not already associated with an LU.

When defining LUs with LUADD parameters in APPCPMxx parmlib members, it is advisable to designate one LU per transaction scheduler as the base LU. When more than one LU is defined as the base LU for a transaction scheduler, the most recently defined LU is the base LU.

If you are running APPC/MVS applications that expect the system base LU to be associated with a transaction scheduler, do not define a NOSCHED LU as the system base LU.

## Naming LUs

---

When naming an MVS local LU, keep in mind that every LU 6.2 node in an SNA network requires a unique name. Not only does an LU 6.2 on MVS require a name different from every other LU 6.2 in the network, but the LU name must be different from names given to other types of SNA nodes, such as PUs, subarea nodes, and system services control points (SSCPs). An LU name consists of two parts: a network-ID portion, which is the 1- through 8-byte ID of the network; and a network-LU-name portion, which is the 1- through 8-byte local LU name. The entire name can be up to 17 bytes in length, in its network-qualified (or fully qualified) form, in which both parts are concatenated by a period: *network\_id.network\_lu\_name*.

Depending on the level of VTAM and APPC/MVS your installation uses, the network-LU-name portion of an LU name must be unique either for every single LU, or only for each LU within one network. The latter case is possible only if your installation's APPC/MVS LUs are enabled to support network-qualified names. See [“Using Network-Qualified Names Support” on page 96](#) for more information about this support, which allows your installation to more easily manage changes to interconnected networks.

In addition, your installation can associate a VTAM generic resource name with APPC/MVS LUs, to improve availability of APPC/MVS resources and, to a certain extent, balance session workload among APPC/MVS LUs. Using VTAM generic resources can also reduce the effort, complexity, and cost of managing a distributed processing environment that includes MVS systems. See [“Assigning a VTAM Generic Resource Name to APPC/MVS LUs” on page 98](#) for more information.

## Using Network-Qualified Names Support

If your installation has VTAM Version 4 Release 4 installed, you can enable APPC/MVS LUs to support of VTAM network-qualified LU names, which reduces the effort of changing the distributed processing environment in an installation that includes several interconnected networks. Previously, if your company merged with another, and added that company's existing network to your installation, system programmers or administrators had to rename any LU that did not have a unique network LU name (the 8-byte local LU name) within the larger installation. Renaming an LU requires changes to several sources of configuration data on several systems, which complicates the tasks required only to define an LU on other systems.

With APPC/MVS support of network-qualified names, renaming LUs is no longer necessary when your installation adds networks containing z/OS systems. To use this support, you specify the NQN parameter on the LUADD statement to enable that APPC/MVS LU to use the entire network-qualified name for partner LUs. (Without this support, APPC/MVS uses only the network-LU-name portion of an LU name on

outbound Allocate calls. In this case, the results of Allocate calls using a network-qualified partner LU name are not guaranteed to be established with the correct partner LU.)

Once your installation changes LUADD statements to make LUs capable of using the entire LU name, the network ID makes the LU name unique. Then, your installation's TPs can use network-qualified partner LU names on Allocate calls, with guaranteed results.

## Deciding When to Use Network-Qualified Names

With support for network-qualified names, system management becomes easier for installations with frequently changing network configurations. Regardless of the frequency of network changes, consider enabling APPC/MVS LUs to use network-qualified names if Allocate calls in TPs or side information for your installation's systems use such names.

If your installation's configuration is relatively stable, however, simpler system management might not be worth the effects of the following security restrictions:

- Security management is limited to defining LU-to-LU access authority through RACF APPCLU profiles. RACF APPL and APPCPORT profiles do not support network-qualified LU names, so your installation must have unique network LU names for each LU to reliably:
  - Limit access to a specific local LU from a specific partner LU, or
  - Limit access to the local system from a specific partner LU.
- Security management for persistent verification is unpredictable as well, because only the network-LU-name portion of an LU name is used to verify persistent verification requests. Without unique network LU names, LUs might, for example, accept conversations from the wrong partner LUs.

Because of this unpredictability, IBM does not recommend the use of APPC/MVS support for network-qualified names for those LUs that your installation uses for persistent verification requests.

## Defining LUs to Support Network-Qualified Names

When you decide to allow APPC/MVS LUs to support network-qualified names, make sure you have VTAM V4R4 installed, and do the following:

1. If your installation uses a security product to define LU-to-LU access authority, changes are required for those security profiles. For example, if a RACF APPCLU class defines LU-to-LU access authority, make sure existing or new APPCLU profiles use the network-qualified name for both the local LU and partner LU names. In other words, for each LU to be enabled to use network-qualified names, all APPCLU profiles currently defined for that LU in the form "local-LU-network-id.local-LU-name.partner-LU-name" must have a corresponding profile in the form "local-LU-network-id.local-LU-name.partner-LU-network-id.partner-LU-name").

You cannot specify network-qualified LU names on any other type of RACF profiles. For more information about APPCLU changes, see [“Defining LU-to-LU Access Authority with RACF APPCLU Profiles” on page 139](#).

2. Through an APPCPMxx parmlib member, use an LUDEL statement to delete each existing, active LU that is to support network-qualified names.
3. Also through an APPCPMxx member, use an LUADD statement with the NQN parameter for each LU that is to be enabled to support network-qualified LU names for its partner LUs.

Changes to existing TPs are not necessary; however, consider notifying your installation's application programmers of the change in APPC/MVS processing of network-qualified partner LU names, so they can determine what changes, if any, might be necessary for the TPs they design and maintain.

## Displaying APPC/MVS Information

With APPC/MVS support for network-qualified names, you can request the following information for APPC/MVS TPs or LUs through the DISPLAY APPC command:

- All partner LUs in only the specified network

- All the partner LUs that share the same specified network\_LU\_name in all the networks in the installation
- Only the partner LU that has a network-qualified name that matches the specified network\_ID and network\_LU\_name
- All partner LUs in all networks.

The values for the PNET parameter, together with those for the PLUN parameter, determine the information displayed. See “Tracking Changes to the APPC/MVS Configuration and Workload” on page 196 for examples of DISPLAY output; See *z/OS MVS System Commands* for the syntax and parameter descriptions of the DISPLAY APPC command.

## Assigning a VTAM Generic Resource Name to APPC/MVS LUs

If your installation has VTAM Version 4 Release 4 installed, you can use APPC/MVS support of VTAM generic resources to:

- Improve availability of APPC/MVS resources. If one LU in the generic resource group or one z/OS system is brought down or fails, APPC/MVS work can continue because other group members are still available to handle requests that specify the generic resource name. Work from remote systems is less affected by the removal of any single APPC/MVS LU or z/OS system. Additionally, changes in system configuration, capacity, and maintenance have less effect on APPC/MVS work.
- Provide a single-system image for a multi-system APPC/MVS configuration. With generic resource names, transaction programs (TPs) from remote systems can establish conversations with APPC/MVS partner TPs on any z/OS system; programmers do not need to know specific partner LU names or need to update TPs whenever the APPC/MVS configuration changes. Note that APPC/MVS TPs can use generic resource names only for partner LUs, not for local LUs.
- More easily expand the APPC/MVS configuration. Additional APPC/MVS LUs associated with the same generic resource name can provide immediate improvement in performance and availability, with few or no required changes for APPC/MVS TPs or side information.
- Distribute work among two or more active APPC/MVS LUs on a single MVS system or in a sysplex, so that each LU is used as efficiently as possible. VTAM and WLM distribute session workload among members of a generic resource group, thus reducing contention for specific LUs, and improving performance of systems and TPs.

Perhaps the simplest, most efficient method of using VTAM generic resources is to copy an existing APPC/MVS LU's definitions for use on the same or additional z/OS systems. Before using this method, however, you need to decide which LUs belong in a generic resource group.

## Deciding which APPC/MVS LUs should be Members of a Generic Resource Group

Although the use of generic resource names for APPC/MVS LUs can benefit your installation, the mix of transaction programs that your installation uses, and the complexity of the APPC/MVS configuration, affect how easily you achieve those benefits. For example, your installation's current set-up might include many TPs that are handled by one APPC/MVS LU on a single z/OS system in a sysplex. With generic resource support, it is relatively easy to distribute this work among additional LUs on the same z/OS system, or among additional LUs on other systems in the sysplex. To do so requires the following steps:

1. With LUADD statements in parmlib member APPCPMxx, define the additional LUs, using the original LU's specific name as the generic resource name for the group.
2. Modify the LUADD statement for the original LU, using its original name as the generic resource name, and supplying a new specific name.

No changes to the TPs are necessary.

For another example, suppose your installation has multiple LUs in the sysplex that process the same work. In this case, using one generic resource name for the group of LUs requires more work than the preceding example:



1. Use the specific name of one of the LUs as the generic resource name for the group.
2. Modify the LUADD statement for each LU, adding the generic resource name. For the one LU, supply a new specific name as well.
3. Modify the TPs and side information for these LUs, replacing specific LU names with the generic resource name for the group.

Your installation may define more than one generic resource group in its APPC/MVS configuration, but not all APPC/MVS LUs have to belong to a group. Those LUs in a group, however, must have access to all the same resources; they should:

- Have the same, if any, transaction scheduler associated with them.
- Use the same TP profile data set, or use TP profile data sets that contain the same TPs.

For the TPs running on LUs in a generic resource group, make sure that **all** the TPs specify the generic name of the MVS LU on the Allocate request. Otherwise, if you allow some TPs to use the generic name, and others to use the specific name, timing becomes the key to successful allocation requests, even for TPs that ran reliably before.

For example, suppose your installation has set up a configuration like the one in [Figure 62 on page 99](#), which shows two TPs that run on the workstation:

- Timecard logs the hours an employee works; it allocates a conversation with a specific APPC/MVS LU.
- E-Mail retrieves mail from the host system; it allocates a conversation with an APPC/MVS LU by using a generic name

On the host, two APPC/MVS LUs share a generic resource name: LUA and LUB are both known by the generic name MVSLU.

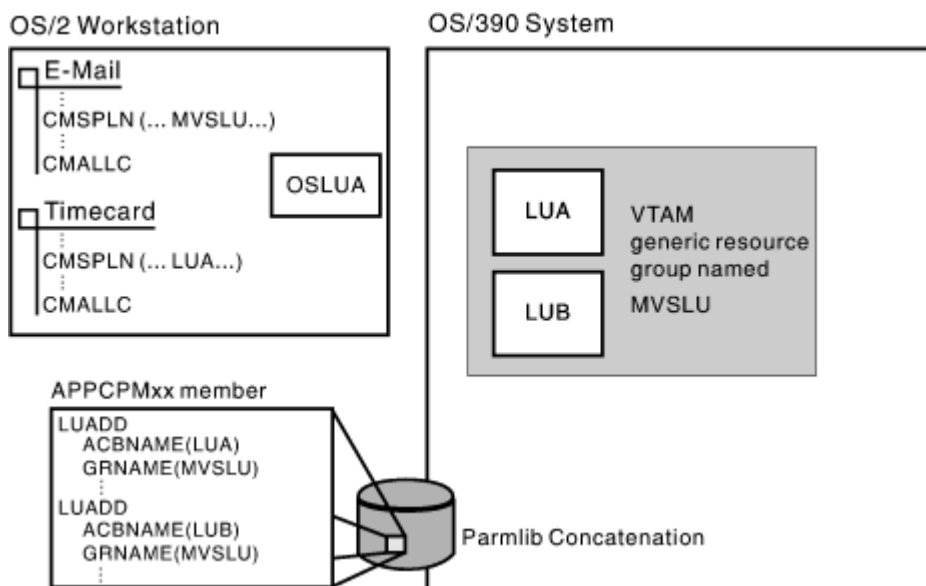


Figure 62. APPC/MVS Configuration with a Mix of Allocate Requests to Specific and Generic LU Names

To understand the problem with allowing the use of a mix of specific and generic LU names on Allocate requests, consider the following sequence of events:

1. On Monday, John arrives at work and invokes Timecard to enter the hours he worked on Friday and over the weekend. Under the covers, a session is established between the workstation LU and the LU specifically requested on the Allocate request, LUA.
2. While checking his recollection of the hours he worked, John also invokes E-Mail to receive his mail. Under the covers, a session is established between the workstation LU and one of the LUs in the generic group. LUA is not available, because a session is already established between the workstation LU and LUA, under its specific name; so LUB is selected for the E-Mail Allocate request. So both conversations are successfully allocated.

3. On Tuesday, John is worried about input that he is expecting from a co-worker, so he invokes E-Mail as soon as he arrives at work. A session is established between the workstation LU and one of the LUs in the generic group— this time, LUA is selected.
4. While waiting for his mail to arrive, John invokes Timecard to enter the hours he worked on Monday. Today, Timecard fails, not because of an error in its own processing, but simply because it wasn't the first TP to allocate to LUA from this workstation LU. Once the workstation LU establishes a session with LUA using LUA's generic name, the workstation LU cannot allocate to LUA using the specific name, while the first session is still bound.

Other timing problems might result because, with generic resource groups, the system fails the Allocate request when both of the following are true:

- Another TP has already successfully issued an allocation request using the specific LU name for an LU in the generic resource group
- No other LUs in the generic resource group are available.

To prevent such failures because of timing, make sure you assign several LUs to the same group, and make sure all the TPs that run on LUs in the group use the generic resource name on the Allocate request. To reduce the amount of work required for the latter step, use an existing, specific LU name as the name for the generic resource group.

### ***Distributing Session Workload***

Your installation's goals for workload distribution also might affect how you define APPC/MVS LUs as VTAM generic resources. Together with the MVS workload manager (WLM), VTAM can balance session (not conversation) workload. For session allocation to a generic resource, VTAM binds the session to one LU in the generic resource group, balancing sessions among the members of the group based on workload information from WLM, if possible, or on session counts. Because VTAM balances sessions only when they are bound, if a particular LU or system becomes constrained, the work running on that LU cannot be redistributed until the sessions are unbound and re-established.

To more closely achieve conversation-level workload balancing, your installation can define LUs as limited resources, as well as defining them as members of a generic resource group. If the LUs are defined as limited resources, VTAM can terminate sessions between LUs if those sessions are not active for an installation-defined time period. For more information about terminating idle sessions, see [\*z/OS Communications Server: SNA Network Implementation Guide\*](#).

As you try to decide which APPC/MVS LUs to assign to a generic resource group, or to define as limited resources, also keep the following points in mind:

- When an APPC/MVS TP allocates a conversation from a local LU that is a member of a generic resource group, and other group members reside on the same system, VTAM balances sessions only within the local system, as long as eligible group members are available. If more than one eligible member is available, VTAM selects the one with less work. Depending on your installation's configuration, this selection might constrain a particular system, if other eligible members of a generic resource group reside on different systems in the sysplex.
- Session workload balancing also varies for generic resource LUs that are associated with a specific transaction scheduler. For these LUs, workload balancing varies depending on the transaction scheduler associated with the LU:
  - Inbound sessions to ASCH or NOSCHED LUs are balanced by analysis of two factors: the number of sessions bound and the capacity of each LU's system
  - Inbound sessions to IMS V5R1 LUs are balanced by the same factors as those for ASCH and NOSCHED LUs, but also by analysis of the actual performance of each LU compared to the performance goals of the transactions it runs.
  - How VTAM balances inbound sessions to alternate transaction schedulers depends on how the alternate transaction scheduler notifies WLM of the completion of the scheduler's transactions.

See [\*z/OS MVS Planning: Workload Management\*](#) for more information about workload balancing.

## ***The Simplest Approach®***

Perhaps the easiest way to assign APPC/MVS LUs to a generic resource group is to:

1. Review the LUADD statements of existing APPC/MVS LUs to:
  - Determine which would serve as good models for members of a specific group, and
  - Decide how many members you want to have in each group, and on what z/OS system each member will reside.

As you make these decisions, remember the sequence of events in [Figure 62 on page 99](#), and the points in [“Distributing Session Workload” on page 100](#).

2. For each model LU, consider "swapping" the real name for the generic resource name; that is:
  - Update the LUADD statement to use the specific name as the generic resource name, which is specified on the GRNAME parameter, and
  - Define a new specific name for the LU.

To avoid potential problems, make sure the generic resource name is unique within a single network. Within a single system, VTAM does not allow the use of the same name for a USERVAR and a generic resource group. Within a single network, VTAM does not prohibit the use of a generic resource name that is the same as a USERVAR, alias, or real LU name, but such duplication might cause undesired results for Allocation requests, or might make problem determination more difficult. For example, the use of USERVARs might cause similar timing problems as described in [Step “4” on page 100](#); once a local LU establishes a session with a partner LU, using a USERVAR for the partner LU name, subsequent Allocate requests with the partner's real LU name will fail. See [z/OS Communications Server: SNA Network Implementation Guide](#) for more information about USERVARs and aliases.

3. Make a copy of the updated LUADD statement for each additional member in the group, and change the specific name in each copy.
4. Use those copies in the appropriate APPCPMxx parmlib member for the z/OS systems on which the group members will reside.

With this approach, the migration effort is reduced to changing only LU definitions in parmlib, rather than changing those definitions **and** all the Allocate calls issued by TPs that run at your installation. Allocate calls do not necessarily need to be changed, because the names they specify for partner LU now might be the generic resource group names.

See [“Defining LUs to a Generic Resource Group” on page 102](#) for the procedure for assigning generic resource names for APPC/MVS LUs.

## ***Alternative Configurations***

To reduce the effort required and potential problems, try to simplify your installation's APPC/MVS configuration by following these guidelines:

- One is a lonely number: Isolate those specific LUs.

If you have to allow certain TPs to issue Allocate requests with specific LUs, do not make those LUs part of a generic resource group. For example, in [Figure 62 on page 99](#), removing LUA from the generic group MVSLU would allow both E-Mail and Timecard to successfully allocate conversations regardless of the order in which they are invoked.

- Two is not as bad as one: Have **at least two active LUs** in each generic group.

Using the configuration in [Figure 62 on page 99](#) as an example again, note that, if LUB fails, LUA is the only LU available to accept work allocated with the generic name MVSLU. In this case, the failure of Timecard could occur more frequently, because LUA could be the only available choice in the generic group. If you add a third LU to the generic group MVSLU, you can reduce the possibility of allocation failures.

## Defining LUs to a Generic Resource Group

To register APPC/MVS LUs as members of a generic resource group, your installation must:

1. Install VTAM Version 4 Release 4 or later, and meet the requirements documented for generic resources in [z/OS Communications Server: SNA Network Implementation Guide](#).

If an earlier level of VTAM is installed, and a generic resource name is specified on the LUADD statement for an LU, APPC/MVS activates the LU, but without using the generic resource name. In this case, Allocate requests specifying the generic resource name will fail.

2. Select an appropriate generic resource name. Refer to [“The Simplest Approach” on page 101](#) for guidelines for selecting names.
3. Through the appropriate commands for the installation's security product, protect LUs in a generic resource group by:
  - Defining LU-to-LU access authority (APPCLU profiles)
  - Defining which user IDs are authorized to allocate conversations with those LUs (APPL profiles)
  - Prohibiting non-APF-authorized programs from registering with a generic resource name (VTAMAPPL profiles).

If the LUs in the same generic resource group reside on different MVS systems, some of these commands must be entered on each system on which the LU members reside. You cannot specify generic resource names on any other security profiles. Refer to the appropriate sections of Chapter 10, [“Setting up Network Security,” on page 133](#) for details about using RACF commands to protect LUs in a generic resource group.

4. Through an APPCPMxx parmlib member, use an LUDEL statement to delete each existing, active LU that is to register with a generic resource name.
5. Also through an APPCPMxx member, use an LUADD statement with the GRNAME parameter to define each LU and associate it with a generic resource name.

After these requirements are met, the installation must complete the following steps, if the generic resource name chosen was not the specific name of an existing LU. Until these steps are completed, the installation cannot benefit from using generic resource names for APPC/MVS LUs.

- Replace specific partner LU names with a generic resource name in side information. Do not use a generic resource name in side information entries that APPC/MVS servers use, or an error results.
- Modify existing, or coding new, TPs to specify the generic resource name for the partner LU on calls to the CPI-C CMSPLN verb, or any version of the APPC/MVS Allocate callable service.

## Displaying APPC/MVS Information for LUs in a Generic Resource Group

To determine which LUs registered as part of the same generic resource group, issue the VTAM DISPLAY ID command, specifying a generic resource name. The resulting display lists the real name for each LU. This command also is useful if you do not know an LU's real name but you want to use the LU name as a filter on a DISPLAY APPC command for LU, TP, or server information.

To determine the generic resource name with which a specific LU registered, issue DISPLAY APPC,LU with a specific LU name. The resulting display lists the LU's generic resource name.

If necessary, see the following for more information:

- [z/OS Communications Server: SNA Operation](#) for syntax and parameter descriptions of the DISPLAY ID command.
- [z/OS MVS System Commands](#) for syntax and parameter descriptions of the DISPLAY APPC command.
- [“Tracking Changes to the APPC/MVS Configuration and Workload” on page 196](#) for examples of DISPLAY APPC command output.

## Diagnosing Errors Involving LUs in a Generic Resource Group

APPC/MVS issues ATBxxxI messages to indicate the following errors:

- Incorrect syntax or other errors for the GRNAME parameter on the LUADD statement, including multiple GRNAME parameters on the same LUADD, incorrect characters in the value for GRNAME, and so on. See [z/OS MVS Initialization and Tuning Reference](#) for coding details for the GRNAME parameter.
- An attempt to dynamically change the value of the GRNAME parameter. To change the value, you must:
  1. Delete the LU by entering a SET APPC command that specifies an APPCPMxx parmlib member containing an LUDEL statement
  2. Alter the GRNAME value on an LUADD statement
  3. Add the LU by entering a SET APPC command that specifies an APPCPMxx parmlib member containing the LUADD statement.
- Incorrect level of VTAM is installed. Unless your installation is using VTAM V4R4 or later, APPC/MVS activates the LU, but without the association with the generic resource name. Allocate requests that specify the generic resource name will fail.
- VTAM rejects the generic resource name. In this case, APPC/MVS cannot continue processing the LUADD statement, so the LU is not activated. Use the VTAM return and reason codes in message ATB066I or ATB067I to determine the cause of the error.

If necessary, see [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#) for descriptions of the messages APPC/MVS issues for errors related to the use of VTAM generic resource names.

APPC/MVS also reports an error through a reason code for the Register\_For\_Allocates service when an APPC/MVS server uses a generic resource name explicitly on that service call, or implicitly through a side information entry. The descriptions of the Register\_For\_Allocates parameters Local\_LU\_name and Sym\_dest\_name, in [z/OS MVS Programming: Writing Servers for APPC/MVS](#), contain more details about this error condition.

If an APPC/MVS LU is a member of a generic resource group, its generic resource name appears in the following diagnostic data:

- CONFIGURATION report output for the IPCS APPCDATA subcommand, which is described in [z/OS MVS Diagnosis: Reference](#).
- Application programming interface (API) trace data, if the LU is being traced. More information about the API trace facility and its output appears in [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

## Setting Up a Session for APPC/MVS

---

Sessions determine where a conversation can flow and the way it flows. An installation can set up an initial session to establish defaults for APPC/MVS sessions, or it can set up specific sessions with unique logon modes, session limits, and contention requirements.

To set up an initial session that accesses the SNA network and establishes defaults:

1. Define a local LU to APPC/MVS
2. Define an APPC logon mode in the VTAM logon mode table, if you want to define session characteristics other than those for the VTAM default mode
3. Define the local LU to VTAM with an APPL definition statement that also names the logon mode table containing the APPC logon mode
4. Define a partner LU on the same z/OS system or on a peer system
5. Provide the logon mode name to applications that specify it on the Allocate call, if you define an APPC logon mode instead of using the VTAM default mode.

Each of these steps is described in more detail in this section. For information about setting up specific sessions with unique characteristics, see [“Customizing Sessions for APPC/MVS” on page 110](#).

## Defining a Local LU on MVS

An APPC/MVS local LU is defined in MVS by an LUADD statement in an APPCPMxx parmlib member. The LUADD names the local LU, associates it with a transaction scheduler and a TP profile file, and assigns it a level of TP to process.

The following is an example of an LUADD statement that defines an LU named MVSLU01. MVSLU01 is the base LU for the APPC/MVS transaction scheduler, has SYS1.APPCTP for a TP profile file, is a USER level LU, which means that it processes all levels (SYSTEM, GROUP, USER) of incoming requests that enter the LU, and has sessions that persist for 3600 seconds following an interruption.

```
LUADD
  ACBNAME(MVSLU01)
  SCHED(ASCH)
  BASE
  TPDATA(SYS1.APPCTP)
  TPLEVEL(USER)
  PSTIMER(3600)
```

Figure 63. Example of an LUADD Statement

For more information about an LUADD statement, see [Chapter 9, “Controlling Configuration through APPCPMxx,”](#) on page 121.

## Defining an APPC Logon Mode

A logon mode contains a set of parameters and protocols that determines the communication characteristics of a session. Logon modes are entries in a logon mode table, a compiled version of which exists in SYS1.VTAMLIB.

To enable LU 6.2 on MVS, you need the VTAM logon mode SNASVCMG that is in the SYS1.SAMPLIB logon mode table sample named ISTINCLM.

Further requirements depend on the version of VTAM your installation is using:

- With VTAM 4.3 or earlier, you must define at least one logon mode entry other than SNASVCMG. To define additional logon modes, you may copy SNASVCMG, rename it and, if desired, alter the values.
- With VTAM 4.4 or later, you may define additional logon modes, but are not required to do so. However, consider defining additional modes, to ensure that the system provides the session characteristics that are appropriate for your installation's APPC/MVS work. If you do not provide any logon modes other than SNASVCMG, APPC/MVS uses logon mode ATB#MODE, which means that session characteristics are determined by the value specified, or by the default, for the DLOGMOD parameter on the VTAM APPL statement.

For an explanation of the parameters in SNASVCMG, and a description of the DLOGMOD parameter of the APPL statement, see [z/OS Communications Server: SNA Resource Definition Reference](#).

Figure 64 on page 104 shows a logon mode that controls session level pacing. Controlling pacing is especially important for sessions between unlike systems that have differing processing capabilities.

```
*****
*          LOGON MODE TABLE ENTRY FOR PC SESSIONS          *
*****
APPCPCLM MODEENT
      LOGMODE=APPCPCLM,                                     X
      RUSIZES=X'8787',                                       X
      SRCVPAC=X'00',                                         X
      SSNDPAC=X'01'
```

Figure 64. Example logon mode (APPCPCLM)



The maximum length of data that can be sent over a session is specified by the RUSIZES parameter. VTAM formats data into RUs and sends those RUs across the network. Pacing parameter SSNDPAC controls the number of RUs sent before the sending VTAM waits for a response from the receiver. When the receiver cannot accept the RUs fast enough, VTAM buffers become flooded with data. By specifying the appropriate pacing parameters, you can prevent the flooding of buffers.

Figure 65 on page 105 shows a logon mode that enables dependent LU support.

```
*****
*      LOGON MODE TABLE ENTRY FOR PC SESSIONS, DEPENDENT LU      *
*****
MVSAPPC MODEENT LOGMODE=MVSAPPC,      APPC/MVS SESSION - DEPENDENT LU  X
                TYPE=0,                NEGOTIATED BIND                  X
                FMPROF=X'13',           X                               X
                TSPROF=X'07',           X                               X
                PRIPROT=X'B0',           EX/DEF RESPONSE                X
                PRIPROT=X'B0',           EX/DEF RESPONSE                X
                SECPROT=X'B0',           EX/DEF RESPONSE                X
                COMPROT=X'50A0',         X                               X
                SSNDPAC=X'01',           X                               X
                SRCVPAC=X'00',           X                               X
                RUSIZES=X'8989',         4096 BYTE MAX RU SIZE BOTH DIR X
                PSERVIC=X'0602000000000000000000000000' LU TYPE 6.2
```

Figure 65. Example logon mode (MVSAPPC)

In Figure 65 on page 105, important parameters to note are:

#### LOGMODE

Specifies the logon mode name to be used as a key for the session parameters in this table entry. This logon mode name corresponds to the logon mode an application programmer specifies in side information or in an APPC/MVS Allocate call.

#### RUSIZES

Specifies the maximum length of data in bytes that can be sent. The suggested value of X'8989' translates to a maximum length of 4096 bytes of data that can be sent at a time from each direction. There is no limit on how much total information can be sent.

#### SRCVPAC

Specifies the secondary receive pacing count. The suggested value is X'00'. If zero, the value of the VPACING operand on the VTAM APPL statement controls both send and receive pacing for all sessions in all modes. A value of zero makes it easier to predict pacing results and makes it easier to maintain pacing definitions.

If non-zero, the VPACING value controls pacing in one direction, and the SRCVPAC value controls it in the other. LU 6.2 protocols make it difficult to predict which parameter will be in control at any given time.

#### SSNDPAC

Specifies the secondary send pacing count. Do not specify zero. If zero is used, outbound pacing for sessions is disabled, which can result in problems with IOBUF storage.

For more detailed information about logon mode parameters, see [z/OS Communications Server: SNA Resource Definition Reference](#).

## Defining the Local LU to VTAM

When VTAM is initialized, LUs are activated based on information contained in SYS1.VTAMLST. Therefore, all APPC/MVS LUs must be defined to both VTAM and MVS.

An APPC/MVS local LU is defined to VTAM with a VTAM application (APPL) definition statement in SYS1.VTAMLST. The APPL statement:

- Names the MVS local LU and identifies it as type 6.2.
- Sets up defaults for the LU's sessions.

- Specifies the name of the logon mode table that contains logon modes used by the LU.
- Defines security for the LU.

The ATBAPPL member of SYS1.SAMPLIB contains an example that shows how a VTAM APPL statement might be coded for an APPC/MVS local LU named MVSLU01. This example includes some of the VTAM APPL statement parameters that are briefly described in [“VTAM APPL Statement Parameters”](#) on page 106. To use the sample APPL contained in the ATBAPPL member of SYS1.SAMPLIB, you need the following additional members of SYS1.SAMPLIB:

#### **ATBLMODE**

Sample source information used to define a VTAM logon mode table. ATBLMODE defines the following logon modes:

##### **APPCHOST**

Table entry for host target

##### **APPCPCLM**

Table entry for PC target (see the example shown in [Figure 64](#) on page 104)

##### **SNASVCMG**

Table entry for resources capable of acting as LU 6.2 devices, required for LU management (see the example in SYS1.SAMPLIB member ISTINCLM)

#### **ATBLJOB**

Sample JCL that assembles and links the sample VTAM logon mode table ATBLMODE. The MODETAB statement in this file must match the MODETAB statement in your APPL.

## **VTAM APPL Statement Parameters**

The following list of APPL statement parameters briefly describes those parameters that you can use for APPC/MVS LUs. Some parameters are used in the example in SYS1.SAMPLIB member ATBAPPL, which you can copy and modify for your installation's use. Whether you use the example in member ATBAPPL or code APPL statements for APPC/MVS LUs from scratch, refer to:

- [z/OS Communications Server: SNA Network Implementation Guide](#) for detailed guidance about APPLs, and
- [z/OS Communications Server: SNA Resource Definition Reference](#) for complete descriptions of APPL statement syntax and parameters.

#### **name**

Assigns a name to the local LU. In APPC/MVS, this APPL entry name must match the value specified on the ACBNAME parameter. This name must be unique within all interconnected networks, unless the LU will be receiving Allocate requests from partner LUs that are enabled to support network-qualified names. (An APPC/MVS LU is enabled for this support when its LUADD statement in parmlib member APPCPMxx contains the NQN parameter).

#### **ACBNAME=LUname**

Specifies the name of the APPC/MVS local LU as it appears in the LUADD parmlib statement of an APPCPMxx parmlib member. This name is specified on the access method control block (ACB) that APPC/MVS uses to identify the LU to VTAM (through the VTAM OPEN macro). In APPC/MVS, this value must match the APPL entry name (that is, the value in the name or label field of the APPL statement). If left blank, it defaults to the entry name.

#### **APPC=YES**

Permits APPC/MVS to use the APPC/VTAM functions. This value is required for all APPC/MVS LUs.

#### **ATNLOSS=value**

Specifies whether the ATTN exit for APPC/MVS should be scheduled for session deactivations. You must specify ATNLOSS=ALL for each APPC/MVS LU that is to be enabled for protected conversation support.



**AUTOSES=*n***

Specifies the number of contention-winner sessions that VTAM is to activate automatically before APPC/MVS requests a conversation. VTAM establishes the specified number of sessions when APPC/MVS issues the first Change Number of Sessions (CNOS) to APPC/VTAM for a particular partner LU and logon mode combination.

**DMINWNL=*n***

Specifies the default minimum number of parallel sessions for which VTAM will negotiate for the local LU to be the contention winner.

**DMINWNR=*n***

Specifies the default minimum number of parallel sessions for which VTAM will negotiate for the partner LU to be the contention winner.

**DDRAINL=*NALLOW***

Specifies whether the LU is allowed to drain waiting allocation requests when the session limits have been set to zero. In this case, APPC/MVS does not support draining; therefore, DDRAINL must be coded with a value of *NALLOW*.

**DRESPL=*NALLOW***

Specifies whether the local LU accepts responsibility for deactivating sessions upon receipt of a CNOS request that names the local LU as the responsible LU. For all APPC/MVS LUs, DRESPL must be coded with a value of *NALLOW* so that VTAM will assign the responsibility for deactivating sessions to the partner LU that sent the CNOS request.

**DSESLIM=*n***

Specifies the default maximum number of sessions to be allowed between the local LU and a partner LU on a given logon mode.

**EAS=*n***

Sets the approximate number of concurrent sessions available from this LU.

**MODETAB=*logon mode table name***

Specifies the name of the VTAM logon mode table. This table is composed of a series of entries, each of which defines a set of session parameters for a particular logon mode name.

**SECACPT=*NONE|CONV|ALREADYV|PERSISTV|AVPV***

Specifies default FMH5 access security information acceptance.

**SRBEXIT=*YES***

Allows VTAM exits to be in branch-entered SRB mode, supervisor state, and key zero. For APPC/MVS's VTAM exits to execute in the correct environment, this value must be *YES*.

**SYNCLVL=*value***

Specifies the synchronization level that the APPC/MVS LU supports. SYNCLVL=SYNCPT is required for each APPC/MVS LU that is to be enabled for protected conversation support.

**VERIFY=*OPTIONAL|REQUIRED***

Specifies whether VTAM performs session-level LU-to-LU verification during session activation.

**VPACING=*n***

Specifies the maximum number of messages that another system can send to the APPC/MVS local LU during a conversation before waiting to receive a pacing response.

## APPC Configuration in VTAM

APPLs reside in one or more members of the SYS1.VTAMLST system library. Each APPL must also connect to a logon mode table (MODETAB) and class of service table (COSTAB).

When APPC/MVS is initializing, it tries to connect each of its LUs to the network by issuing a VTAM OPEN macro. The OPEN macro tells VTAM to open an ACB corresponding to the LU name. If VTAM is not yet started, or if the OPEN fails, APPC/MVS will periodically issue the OPEN until it succeeds or until an APPCPMxx parmlib member containing an LUDEL statement for that LU is added to the configuration by a SET APPC operator command.

[Figure 66 on page 108](#) shows the relationship between the various pieces of VTAM configuration data.

## VTAM DEFINITION STATEMENT IN SYS1.VTAMLST

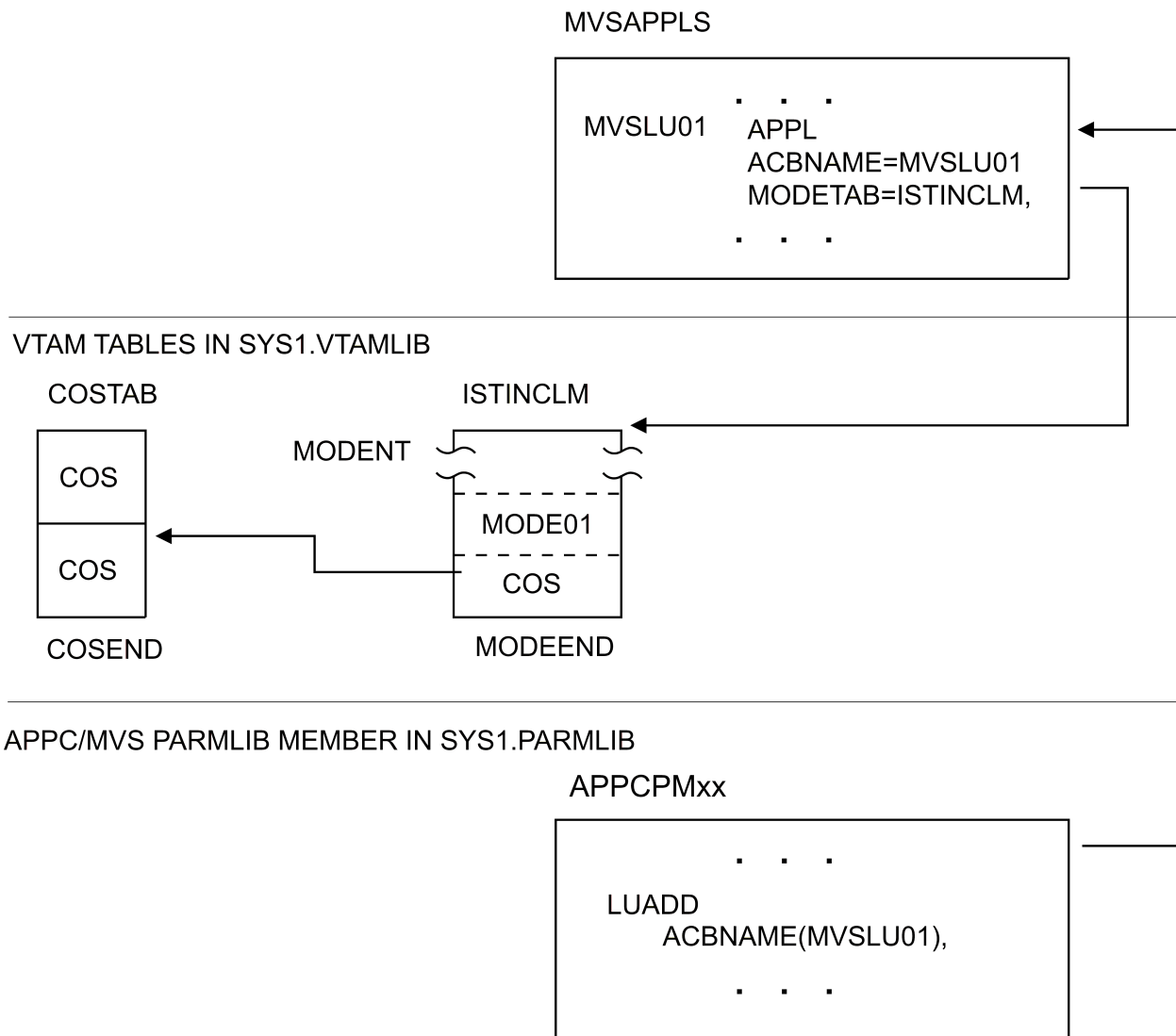


Figure 66. Application Definition to VTAM

## Defining a Partner LU on the Same MVS System

When a conversation exists between two transaction programs on the same z/OS system, the LUs for the conversation are also on the same system. The conversation can use a single LU that acts as both local LU and partner LU, or it can use one LU as the local LU and another as the partner LU.

A conversation that uses a single LU as both local LU and partner LU is an LU=OWN conversation. When a TP allocates the conversation, it can either explicitly request the local LU as the partner LU, or leave the partner LU name blank.

A conversation between two APPC/MVS TPs that uses a local LU and a partner LU on the same system is an LU=LOCAL conversation.

**Note:** Requirements for APPC/MVS LUs, and the behavior of LU=OWN and LU=LOCAL conversations, vary depending on the VTAM release your installation uses.

## Defining a Partner LU on a Peer System

Before an APPC/MVS LU can communicate with a partner across an SNA network, programmers for non-MVS peer systems have certain responsibilities. In general, the major responsibilities for peer system programmers are:

- Define local LUs on the peer systems that APPC/MVS TPs will code as partner LUs in Allocate calls and in side information.
- Define MVS local LUs as partner LUs on the peer system. The name of the MVS LU must be coded as the partner LU name.
- Define logon modes for sessions that are to be bound between the peer system and MVS. The names of the logon modes should match the names used in MVS.
- Make sure that the conversation security levels defined for local LUs allow their partners to send the proper security information. If a TP on the partner LU attempts to allocate a conversation, specifying a security type that sends security fields that the local LU cannot support, the system downgrades the allocate request. In such a case, the local LU receives less security information than the TP was designed to send, so the results might not be what you expected for this conversation.

To check the conversation security levels for APPC/MVS LUs, review either:

- The SECACPT parameter of the VTAM APPL definition statement, **or**
- The CONVSEC field in the SESSION segment of the RACF APPCLU profile, if such a profile exists for this LU. RACF APPCLU profiles override the values in a VTAM APPL definition statement.

For more information, see:

- [“Defining the Local LU to VTAM” on page 105](#), for details about the VTAM APPL statement.
- [Chapter 10, “Setting up Network Security,” on page 133](#), for details about RACF APPCLU class profiles.
- [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#), for descriptions of security fields sent on Allocate requests, depending on the value specified for the Security\_type parameter.

Session requirements for specific systems can be obtained from the following references.

AS/400:

- *AS/400 APPC Programmer's Guide*
- *OS/400 Communications Configuration Reference*

VM:

- *VM/ESA V2R1.0 Connectivity*

## Providing the Logon Mode Name to Applications

An application that uses the APPC/MVS callable services can specify a logon mode in its Allocate call. In the following COBOL example, the Mode\_name parameter is pre-initialized to MODE01 before the Allocate call is issued.

```
MOVE "MODE01" TO MODE_NAME.  
CALL "ATBALC2" USING  
    CONVERSATION_TYPE,  
    SYM_DEST_NAME,  
    PARTNER_LU_NAME,  
    MODE_NAME,  
    ....  
    RETURN_CODE.
```

Figure 67. COBOL example of an ATBALC2 call using a logon mode name

When a symbolic destination name replaces the need for specifying the logon mode, the logon mode must be named in the side information for that symbolic destination name. In the COBOL example that follows, the symbolic destination name `USR3NEWS` is used in the CPI Communications Initialize\_Conversation (`CMINIT`) call, which initializes conversations. The symbolic destination name is resolved by its side information in the example box where the logon mode name `MODE01` is specified.

```
MOVE "USR3NEWS" TO SYM_DEST_NAME.  
CALL "CMINIT" USING CONVERSATION_ID,  
                    SYM_DEST_NAME,  
                    CM_RETCODE.
```

Figure 68. COBOL example of a `CMINIT` call using a symbolic destination name

```
DESTNAME(USR3NEWS)  
MODENAME(MODE01)  
TPNAME(NEWS)  
PARTNER_LU(USER3LU)
```

Figure 69. Side Information for `USR3NEWS`

For more information about providing a logon mode through an Allocate call or side information, see [“Specifying a Logon Mode for a Conversation” on page 111](#).

## Customizing Sessions for APPC/MVS

When sessions are bound between an MVS LU and any partner LU, VTAM determines session characteristics from a combination of:

- VTAM session defaults
- Logon modes as specified in:
  - APPL statements
  - Allocate calls

When more than one parameter refers to the same characteristic, certain parameters override others, as illustrated in Figure 70 on page 110. A logon mode specified in an Allocate call overrides the parameters specified in the APPL statement. Both the logon mode and the APPL statement override VTAM defaults.

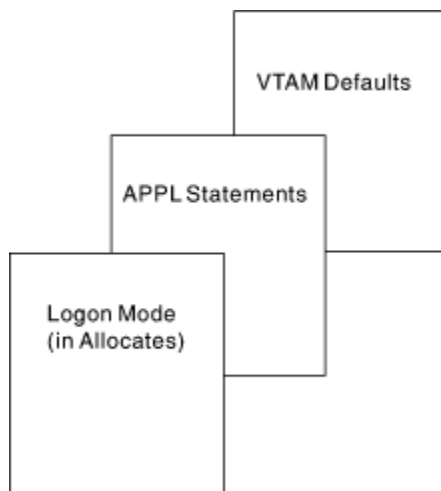


Figure 70. Overriding Session Defaults

When the local LU and its partner have conflicting definitions, VTAM negotiates the differences and resolves them internally.

To customize sessions and override defaults, define additional logon mode entries in the VTAM logon mode table for use in Allocate calls.

## Defining Additional Logon Mode Entries

An installation can create logon modes with varying communication characteristics. These additional logon modes must be entries in a logon mode table contained in SYS1.VTAMLIB. All the logon modes used by an LU should be contained in the table specified in the LU's APPL definition statement.

When an LU uses a number of different logon modes, it might be worthwhile to place those logon modes in a separate logon mode table specifically for the LU. SYS1.SAMPLIB member ATBLMODE contains an example logon mode table with three logon modes, including SNASVCMG, which is required by APPC/MVS.

If you create a new logon mode table, specify the table name after the MODETAB parameter in the LU's APPL statement.

For more information about logon mode tables and logon mode entries, see [z/OS Communications Server: SNA Resource Definition Reference](#). For information about specific session characteristics, such as pacing and controlling the amount of data to pass, see [z/OS Communications Server: SNA Network Implementation Guide](#).

## Specifying a Logon Mode for a Conversation

The Allocate call or side information can specify the logon mode that will control the session characteristics for the conversation. If a number of logon modes with varying communication characteristics have been defined, a programmer has a choice of session types for the TP.

## Using APPC/MVS Protected Conversations Support

To improve data integrity in a distributed processing environment, APPC/MVS, together with RRS, participates in the two-phase commit protocol to provide recovery for transaction programs. The two-phase commit protocol is a set of actions that resource managers and a syncpoint manager perform to ensure that a program's updates to distributed resources are coordinated. Through this protocol, a series of resource updates are treated as an atomic action; that is, the updates are either all made (committed) or not made (backed out).

In z/OS, your installation can enable APPC/MVS logical units (LUs) to act as resource managers. The resources they manage, or protect, are the conversations established between APPC/MVS transaction programs and their partner TPs. To identify their conversations as protected resources, the TPs allocate the conversations with a synchronization level of syncpt. When one of the TPs is ready to commit or back out its changes for a particular unit of work, the TP issues either the Commit or Backout callable service to begin a syncpoint operation. During this operation, the local and partner LUs work with system syncpoint managers to coordinate the changes; RRS is the system syncpoint manager for APPC/MVS LUs.

To allow APPC/MVS TPs and their partner TPs to establish protected conversations, your installation must meet the following requirements:

- Set up and start RRS for resource recovery. For specific RRS requirements, see [z/OS MVS Programming: Resource Recovery](#).
- Install and activate VTAM Version 4 Release 4.
- Set up the APPC/MVS configuration as described in the following topics:
  - [“LU Capability and Mode Name Restrictions” on page 112](#)
  - [“Defining APPC/MVS LUs as Syncpoint Capable” on page 112](#)
  - [“Defining a Log Stream for APPC/MVS” on page 113](#)

- Update existing, or code new, APPC/MVS TPs to allocate protected conversations and request syncpoint services. To do so, refer to:
  - [z/OS MVS Programming: Callable Services for High-Level Languages](#) for general concepts about the two-phase commit protocol and resource recovery, and for coding details for the z/OS commit and backout syncpoint services.
  - [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#) for information about allocating protected conversations, using syncpoint services, and other related updates to the APPC/MVS callable services.

Once your installation has met these requirements, APPC/MVS is enabled to support protected conversations, and to participate in resource recovery. Managing this APPC/MVS configuration and workload requires an understanding of the issues described in [“Managing APPC/MVS Resources for Protected Conversations”](#) on page 116.

## LU Capability and Mode Name Restrictions

APPC/MVS rejects any outbound or inbound requests for protected conversations whenever the partner LU is single-session capable only.

Syncpoint-capable LUs accept both inbound and outbound protected conversations, as long as the mode name used for the Allocate call is a value other than SNASVCMG.

## Defining APPC/MVS LUs as Syncpoint Capable

An installation defines the characteristics and resources for APPC/MVS LUs through APPL definition statements in SYS1.VTAMLST, and LUADD statements in APPCPMxx parmlib members.

To define new, or alter existing, LUs to make them syncpoint capable, complete the steps in the following checklist:

1. Make sure that each LU's APPL definition statement contains the SYNCLVL parameter with a value of SYNCPT. This parameter value defines the LU as capable of accepting conversations with any of the following synchronization levels: syncpt, confirm, or none.
2. Make sure that each LU's APPL definition statement contains the ATNLOSS parameter with a value of ALL.
3. Check LUADD statements to make sure the values match what you want for specific syncpoint-capable LUs. If you want to restrict the LU to process protected conversations only, for example, check the TPDATA parameter to ensure that the TP profile data set is one containing only TPs that allocate protected conversations.

If you need additional information about VTAM resource definitions, refer to:

- [z/OS Communications Server: SNA Network Implementation Guide](#) for detailed guidance about APPLs.
- [z/OS Communications Server: SNA Resource Definition Reference](#) for complete descriptions of APPL statement syntax and parameters.

For additional details about LUADD statements, refer to [“Adding a Local LU — LUADD Statement”](#) on page 122, [“Modifying a Local LU — LUADD Statement”](#) on page 123, and [z/OS MVS Initialization and Tuning Reference](#).

After completing the checklist, the LUs and, by extension, the schedulers and APPC/MVS servers associated with them, are capable of handling protected conversations. Changes to alternate transaction schedulers and APPC/MVS servers are not necessary; however:

- Alternate transaction schedulers may use the new Identify service to have APPC/MVS request a privately managed context for that scheduler. For details, refer to [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#).
- Depending on the model, APPC/MVS servers might require changes to handle protected conversations. For details, refer to [z/OS MVS Programming: Writing Servers for APPC/MVS](#).

## Syncpoint Capabilities Supported by APPC/MVS

This section describes the level of syncpoint support that APPC/MVS supports in relation to the SYNCPT architecture.

An APPC LU negotiates the level of syncpoint support with a partner using the Exchange Log Name (X'1211') GDS Variable. The format of this record is documented in *SNA Formats*. The level of syncpoint support by an LU is expressed in one byte in the exchange log name GDS variable. The following is how APPC sets byte 5 of the Exchange Log Name (X'1211') GDS Variable:

Table 11. How APPC sets byte 5 of the Exchange Log Name (X'1211') GDS Variable		
Bit	Value	Meaning
0	0	Does not support RECOVERY_LEVEL(RESTART)
1	1	The LU name of the creator of the conversation correlator is present in Compare States
2	1	Byte 2 of the PS header contains flags and X'08240001' sense code is accepted in FMH-7
3	0	Does not support presumed abort protocols
4	1	Supports PLNA protocols
5	1	Does not require partner log name validation on a warm start
6	0	Does not support extended capabilities
7	-	Not used for syncpoint capability negotiation

## Defining a Log Stream for APPC/MVS

To provide resource recovery for protected conversations, APPC/MVS requires the names of local and partner LU logs, and the negotiated syncpoint capabilities for each local/partner LU pair. This information needs to be available and accurate for successful resynchronization after a failure. To store this information, APPC/MVS uses a system logger log stream that your installation must set up.

To set up an APPC/MVS log stream, your installation follows the same procedure that is required for other system logger applications:

1. Plan for and set up the APPC/MVS log stream.
2. Plan and set up DASD space needed for APPC/MVS log data sets and staging data sets.
3. Define the APPC/MVS log and staging data sets to global resource serialization in the GRSRNLxx parmlib member.
4. Define authorization to system logger resources for APPC/MVS.
5. Format the LOGR couple data set, identify it to the sysplex, and make it available.
6. Add, update, or delete APPC/MVS policy data (the APPC/MVS log stream and its associated coupling facility structure) in the LOGR policy using the IXCMIAPU utility. APPC/MVS supports DASD ONLY configuration for its log stream in a single system environment.
7. Update the CFRM sysplex couple data set with information on the coupling facility structure associated with the APPC/MVS log stream.
8. Activate the LOGR subsystem.

While doing the setup for an APPC/MVS log stream in a parallel sysplex environment, keep in mind that if an APPC/MVS log stream is defined with DASD ONLY configuration, APPC/MVS from only one system can connect to the log stream. Only APPC/MVS from this one system processes protected conversations. Other systems in a parallel sysplex fail to process protected conversations and issue message ATB203I to document the return and reason codes received from the system logger IXGCONN service.

For more information about these steps, refer to the topic about system logger applications in [z/OS MVS Setting Up a Sysplex](#), **after** reading through the following topics:

- [“Determining the Size of Each Coupling Facility Structure”](#) on page 114
- [“Develop a Naming Convention for System Logger Resources”](#) on page 114
- [“Plan DASD Space for System Logger”](#) on page 115
- [“Managing Log Data: How Much? For How Long?”](#) on page 115
- [“Define Authorization to System Logger Resources”](#) on page 115

These topics contain APPC/MVS-specific details that you need to know before you define APPC/MVS policy data in the LOGR policy. The topic headings match topics in [z/OS MVS Setting Up a Sysplex](#), where you can find general information about LOGR policy data.

## ***Determining the Size of Each Coupling Facility Structure***

Your installation may decide to place the APPC/MVS log stream in its own coupling facility list structure, or in a list structure containing multiple log streams with the same attributes. In either case, as you gather information about the coupling facility structure, and the log streams that map to the structure, note the following APPC-specific information related to parameters in the LOGR policy:

### **AVGBUFSIZE**

The APPC/MVS log stream contains one log block for both of the following:

- Each local/partner LU pair that has established sessions with protected conversations
- Each LU pair, if any, that has outstanding resynchronization work.

Each log block is the same size: 248 bytes. Use 248 as the value for the average size of APPC/MVS log blocks.

### **MAXBUFSIZE**

APPC/MVS requires a buffer size of at least 65276 bytes. If you use a MAXBUFSIZE value that is less than 65276, APPC/MVS issues message ATB209I and does not allow APPC/MVS LUs to handle any protected conversations, until the buffer size is corrected and the LUs restarted.

### **ResidencyTime**

You may use a relatively low value for residency time, because APPC/MVS writes to the log stream infrequently.

## ***Develop a Naming Convention for System Logger Resources***

As part of developing a naming convention for system logger resources, your installation determines the names of log streams. The APPC/MVS log-stream name will depend on the option chosen in APPC's started procedure.

If you want the capability of having more than one APPC log stream per sysplex, then specify *LOGGING = RRSNAME* on the APPC PROC statement:

```
//APPC PROC APPC=&APPC, LOGGING=RRSNAME...
```

If *RRSNAME* is specified, the APPC log stream name will need to be *'ATBAPPC.LU.gname'*, where *gname* is the RRS *GNAME* as defined in the RRS started procedure definition. If the *gname* keyword is not specified, RRS defaults the log group name to the sysplex name. The RRS group name is a way RRS allows multiple logs to be defined in a sysplex. The *RRSNAME* value tells APPC to have its logs coincide with a particular RRS log group. For further information about RRS's *GNAME*, see [z/OS MVS Programming: Resource Recovery](#).

If you do not want the capability of having more than one APPC log stream per sysplex, then specify *LOGGING = LEGACY* on the APPC PROC statement or simply omit the *LOGGING* keyword completely from the APPC PROC statement:

```
//APPC PROC APPC=&APPC, LOGGING=LEGACY
```



or

```
//APPC PROC APPC=&APPC,...
```

If *LEGACY* is specified (explicitly or by default), then the APPC log stream name will need to be 'ATBAPPC.LU.LOGNAMES'.

Based on the *LOGGING* option specified, APPC will attempt to connect to a log stream with the corresponding name designation selected (either ATBAPPC.LU.*rrsgname* or ATBAPPC.LU.LOGNAMES) when an LU is initialized that is configured to support APPC protected conversations.

The installation is responsible for defining a log stream matching one of these two names, depending on the *LOGGING* option specified. The failure to define the log stream name correctly will result in APPC being unable to allocate or receive any protected conversations using this LU.

#### *Considerations when using the APPC **LOGGING** keyword*

When you are using the APPC *LOGGING* keyword consider the following:

- RRS keeps track of the APPC log stream name when APPC initializes itself with RRS for each LU that is configured to accept protected conversations. Before APPC can change its log stream name, it is necessary to remove the old APPC log stream name from RRS's recollection. In order to accomplish this, it is necessary to cold start RRS. For example, when changing the APPC *LOGGING* option to *LEGACY* (the default value) to *RRSGNAME*, RRS must be cold started first before APPC can be started with this new value.

See *z/OS MVS Programming: Resource Recovery* for more information on deleting and redefining RRS log streams.

- When *LOGGING=RRSGNAME* on the APPC PROC, RRS and APPC will both have the same GNAME as one of the qualifiers in their log stream names. If RRS comes down and is restarted with a different GNAME than in the previous instance of RRS, then APPC will need to be recycled to have its log stream name match the new RRS GNAME.

### **Plan DASD Space for System Logger**

Your installation is **not** required to use staging data sets for APPC/MVS log data. However, if a system or coupling facility failure causes the loss of APPC/MVS log data, warm/cold mismatches between local and partner LUs result. To resolve such mismatches, your installation might have to:

1. Bring down APPC/MVS (see Chapter 11, "Operating APPC/MVS," on page 185 for recommended methods of stopping APPC/MVS work cleanly)
2. Use RRS ISPF panels to remove the expressions of interest that APPC/MVS LUs have in units of recovery For more information, see *z/OS MVS Programming: Resource Recovery*.
3. Restart APPC/MVS.

This manual intervention might be more costly than the potential performance impact of using staging data sets. Because APPC/MVS infrequently writes to its log, the performance impact should be relatively slight; so consider defining the APPC/MVS log stream with STG\_DUPLEX(YES) and DUPLEXMODE(UNCOND).

### **Managing Log Data: How Much? For How Long?**

When your installation defines the APPC log stream, make sure that AUTODELETE=NO and RETPD=0 are specified (or use the AUTODELETE and RETPD parameter defaults which are NO and 0, respectively).

Also, if your installation uses the LIKE keyword when defining the APPC log stream, make sure that the log stream specified on the LIKE keyword is not defined with AUTODELETE=YES and RETPD other than zero.

### **Define Authorization to System Logger Resources**

If you start APPC/MVS on more than one MVS image in a sysplex, each of those MVS images must have access to the APPC/MVS log stream.

## Managing APPC/MVS Resources for Protected Conversations

After your installation sets up RRS and APPC/MVS for resource recovery, installation personnel need to know how to manage this new work. APPC/MVS provides system commands and messages that indicate the status of APPC/MVS workload and configuration, so that installation personnel may:

- Display information related to APPC/MVS participation in resource recovery, for LUs, TPs, and units of recovery (URs)
- Change the APPC/MVS configuration or workload, and track those changes
- Resolve error conditions related to resynchronization processing, when intervention is required to restore protected conversation support.

In addition, several tools allow the installation to collect and view diagnostic information for system errors related to protected conversations.

### Displaying Information

Through variations of the DISPLAY command, operators or system programmers can determine the status of APPC/MVS resources related to protected conversations:

- The DISPLAY APPC,LU command output contains the resource manager name for LUs that are registered with RRS, and indicates whether each LU is syncpoint capable
- The DISPLAY APPC,TP command output contains the following for each of the TP's conversations:
  - The logical work unit identifier (LUWID)
  - An indication of whether the conversation is protected or unprotected
  - An indication of whether a syncpoint operation is in progress.
- The DISPLAY APPC,UR command output contains information about each unit of recovery associated with a protected conversation.

DISPLAY APPC,SERVER command output does not contain information about served TPs' protected conversations. To obtain such information, use the DISPLAY APPC,TP command.

For a summary of DISPLAY APPC commands and various sample commands and output, see [“Tracking Changes to the APPC/MVS Configuration and Workload”](#) on page 196. For DISPLAY APPC command syntax and parameter descriptions, see [z/OS MVS System Commands](#). An installation can also use RRS ISPF panels to display APPC unit of work activity. For more information, see [z/OS MVS Programming: Resource Recovery](#).

### Changing the APPC/MVS Configuration or Workload

You can use a number of MVS and VTAM commands to control APPC/MVS LUs, schedulers, servers, and TPs. Topics in [Chapter 11, “Operating APPC/MVS,”](#) on page 185 describe the effects of these commands on protected conversations, if the effects are different from those on unprotected conversations.

One task related to APPC/MVS operations, however, is worth emphasizing here: When you need to stop an APPC/MVS LU, make sure you first use LUDEL to terminate the LU and quiesce its work. IBM recommends using LUDEL before using any other method that stops one or more LUs, but its use is even more important when your installation has enabled protected conversations support. Unless you first allow an LU to quiesce its work, thereby allowing active protected conversations to end normally, incomplete units of recovery (URs) might result. If so, the system defers resynchronization processing for these incomplete URs, until the LU is re-activated on the same system. This delay might cause the programs associated with the incomplete URs to hang until the LU is restarted.

In addition, if your installation defines a syncpoint-capable LU as a member of a VTAM generic resource group, and an operator issues the CANCEL APPC command before issuing LUDEL first, the system cannot correctly clean up LU session affinity (associated with VTAM generic resources) and the APPC/MVS log stream. This situation might result in unbalanced workload sessions for LUs in a generic resource group, and incorrect log stream contents, once APPC is restarted.

## Resolving Error Conditions

Because resource recovery processing might involve many applications, resource managers, and syncpoint managers, your installation's operators and programmers might have to resolve error conditions resulting from application, system, or network failures. APPC/MVS attempts to resolve error conditions whenever possible, but manual intervention is sometimes required for the following situations. APPC/MVS notifies your installation of these situations by issuing ATB2xx messages, which are described in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

### ***RRS failures or other errors***

If RRS is not available or notifies APPC/MVS of an error condition, APPC/MVS is not able to participate in resource recovery. To minimize the effect of these errors on APPC/MVS work, APPC/MVS allows the LU to become active or continue processing, but does not allow it to accept Allocate requests for protected conversations. In some cases, RRS might notify APPC/MVS that the error has been resolved; then APPC/MVS allows the LU to begin accepting protected conversations again. In other cases, manual intervention, such as a cold start of the LU, might be required to restore protected conversations support.

### ***Logging failures detected during LU initialization or restart***

If the system logger is not available or logger services return non-zero codes, APPC/MVS is not able to use its installation-defined log stream to record information about its resource recovery partners. To minimize the effect of this error on APPC/MVS work, APPC/MVS allows the LU to become active, but does not allow it to accept Allocate requests for protected conversations. When you correct the logging problems, you must restart the LU to enable it to accept protected conversations.

### ***Errors that prevent log name exchange from completing successfully***

As part of participation in resource recovery, an APPC/MVS LU must exchange its log name with its partner resource managers. The LU might detect an error during the exchange log name transaction with the partner. To minimize the effect of these errors on APPC/MVS work, APPC/MVS periodically retries the exchange. Meanwhile, the LU is able to process only unprotected conversations.

Most of these errors require communication between the support groups for the applications involved in the exchange. Consider providing procedures for the installation's operators to follow, to resolve the error. For example, using an incorrect RRS GNAME when starting RRS would cause a log name mismatch to be detected by a partner LU. Ensure that the correct RRS GNAME is used from one RRS warm start to another.

### ***Warm/cold log status mismatch***

If, during an exchange log name transaction, the local LU or partner LU detects a warm/cold log status mismatch, APPC/MVS issues operator message ATB210E. Messages ATB70052I and ATB80129I may be returned to the TP.

The log status mismatch may be caused by:

- The wrong level of log data at the local or partner LU, or
- A cold log start at one of the partners. In this case, you might consider manually forcing some units of work at the warm partner to resolve the problem.

If the cold log status is valid for one of the logical units, and if the warm partner is an APPC/MVS managed logical unit of work, then to resolve the warm/cold mismatch, take one of the following actions against the warm partner (listed in order of increasing potential disruption):

1. Restart the warm APPC/MVS LU after removing all interests for the APPC/MVS LU using the RRS ISPF panels.



**Attention:** Before removing incomplete logical units of work, determine whether the incomplete logical units of work can be manually forced or discarded without resynchronization being performed.

- a. Delete the LU from the APPC/MVS configuration by issuing a SET command for a parmlib member with an LUDEL statement for the LU.
- b. Remove all interests for the cold status partner using the RRS ISPF panels. This prevents APPC/MVS from carrying out any resynchronization to the cold status partner for the removed incomplete logical units of work.

The LU is known to RRS as a resource manager. The resource manager naming convention for APPC/MVS LUs is:

```
ATB.network-qualified-network-name.IBM
```

where *fully-qualified-name* is the fully-qualified name of the local LU. For example, if the fully-qualified name is IBMUSM00.Z0C4AP03, the resource manager name to be specified on the RRS ISPF panels is:

```
ATB.IBMUSM00.Z0C4AP03.IBM
```

For information on using the RRS ISPF panels, see [z/OS MVS Programming: Resource Recovery](#).

- c. Add the LU to the APPC/MVS configuration by issuing a SET command for a parmlib member with an LUADD statement for the LU. The APPC/MVS LU will now restart (however, this will be without incomplete logical units of work).
- d. Attempt to initiate a protected conversation between the affected LUs.

**Important:** The following action affects all protected conversations. Take this action only if the previous action does not resolve the problem.

2. Delete and redefine the APPC/MVS LU log stream. This will erase APPC/MVS's knowledge of *all* partners' log information and syncpoint capabilities, not just the cold status partner affected by the problem. For specific information on adding the log stream correctly, see [“Defining a Log Stream for APPC/MVS” on page 113](#).

**Important:** The following action affects all resource managers, including all syncpoint LUs, and any other resource managers that are registered with RRS. Take this action only if the previous actions do not resolve the problem.

3. Cold start RRS. See [z/OS MVS Programming: Resource Recovery](#).

### Log name mismatch

If, during an exchange log name transaction, the local LU or partner LU detects a log name mismatch, APPC/MVS issues operator message ATB211E. Messages ATB70053I and ATB80130I may be returned to the TP.

The log name mismatch may be caused by:

- The incorrect system log being used on the local or partner system.
- An internal error in APPC/MVS logging or in the logging function of the partner system.

If an incorrect system log caused the problem, consider taking the following steps:

1. Attempt to correct the log name mismatch problem on the partner system using the partner system's local log name mismatch recovery procedures. After the recovery procedures for the partner system have been performed successfully, use the RRS ISPF panels to remove all incomplete interests associated with the affected LU pair. See [z/OS MVS Programming: Resource Recovery](#). After removing incomplete interests, restart the APPC/MVS LU.
2. Cause the APPC/MVS LU to initiate purge log name affinity (PLNA) processing. Both the partner and local systems must support PLNA. No incomplete units of work may exist between the partner LUs, and all work must be quiesced in order for the PLNA transaction to be successful.

To initiate a purge log name affinity transaction from the APPC/MVS LU, do the following:



**Attention:** Before cold starting an LU, determine whether the incomplete logical units of work can be manually forced or discarded without resynchronization being performed.

- a. Delete the LU from the APPC/MVS configuration by issuing a SET command for a parmlib member with an LUDEL statement for the LU.

The LUDEL operation will initiate an PLNA transaction with the partner LU.

- b. Determine if any incomplete logical units of work exist between the affected LUs. To do this, issue a D APPC,UR using the LLUN, PNET and PLUN filtering options to restrict the output to the affected LUs.

If the display shows 0 units of recovery for the affected LUs, there are no incomplete units of work for the affected pair.

If the display shows that incomplete units of work exist between the LU pair,

- c. Remove all interests for the LU using the RRS ISPF panels. The resource manager naming convention for APPC/MVS LUs is:

```
ATB.network-qualified-network-name.IBM
```

where *fully-qualified-name* is the fully-qualified name of the local LU. For example, if the fully-qualified name is IBMUSM00.Z0C4AP03, the resource manager name to be specified on the RRS ISPF panels is:

```
ATB.IBMUSM00.Z0C4AP03.IBM
```

For information on using the RRS ISPF panels, see [z/OS MVS Programming: Resource Recovery](#).

- d. Add the LU to the APPC/MVS configuration by issuing a SET command for a parmlib member with an LUADD statement for the LU. The APPC/MVS LU will now restart (however, this will be without incomplete logical units of work).
- e. If incomplete units of work existed in step “2.b” on page 119 and incomplete units of recovery had to be deleted as part of step “2.c” on page 119, then steps “2.a” on page 119 through “2.d” on page 119 must be repeated.
- f. Attempt to initiate a protected conversation between the affected LUs.

### 3. Cold start the partner LU

**Important:** The following action affects all protected conversations. Take this action only if the previous action does not resolve the problem.

4. Delete and redefine the APPC/MVS LU log stream. This will erase APPC/MVS's knowledge of *all* partners' log information and syncpoint capabilities, not just the cold status partner affected by the problem. For specific information on adding the log stream correctly, see “Defining a Log Stream for APPC/MVS” on page 113.

**Important:** The following action affects all resource managers, including all syncpoint LUs, and any other resource managers that are registered with RRS. Take this action only if the previous actions do not resolve the problem.

5. Cold start RRS. See [z/OS MVS Programming: Resource Recovery](#).

## Obtaining Diagnostic Information

For errors that installations cannot resolve without IBM support, IBM provides:

- A new APPC/MVS component trace option, RR, for tracing events related to APPC/MVS participation in resource recovery for protected conversations. New options for the IPCS CTRACE subcommand filter the resulting trace records. For more information, see the SYSAPPC topic in [z/OS MVS Diagnosis: Tools and Service Aids](#).
- The IPCS APPCDATA subcommand for formatting SVC dumps that accompany the X'EC7' system abends that APPC/MVS issues. The APPCDATA CONFIGURATION and CONVERSATION reports contain information related to protected conversations.

For information about APPCDATA subcommand syntax and parameters, see [z/OS MVS IPCS Commands](#). For sample APPCDATA subcommand output, see [z/OS MVS Diagnosis: Reference](#).



---

## Chapter 9. Controlling Configuration through APPCPMxx

The APPCPMxx member of the parmlib concatenation controls the communication functions for the APPC address space. Values in the parmlib member define local LUs, name administrative VSAM KSDSes, and optionally modify session characteristics between local LUs and partner LUs.

References:

[\*z/OS MVS Initialization and Tuning Reference\*](#)

[\*z/OS MVS Planning: Operations\*](#)

[\*z/OS MVS System Messages, Vol 3 \(ASB-BPX\)\*](#)

---

### APPCPMxx Parmlib Member

The APPCPMxx parmlib member contains a combination of three statement types that define or modify the configuration of APPC/MVS LUs. The statement types are:

#### **LUADD**

Defines a local LU for the APPC/MVS configuration.

#### **LUDEL**

Deletes a local LU and its defined sessions from the APPC/MVS configuration.

#### **SIDEINFO**

Specifies the VSAM KSDS name that contains side information for an installation.

### Changing Values

An installation can control its APPC/MVS configuration with different versions of the APPCPMxx parmlib member. One member might contain start-up values and other members contain customized values. An APPCPMxx parmlib member can contain statements that delete previous statements (LUDEL deletes a previous LUADD) or the parmlib member can reissue statements with new parameter values that modify previous statements.

Examples of parmlib members used to delete and modify configurations appear throughout this chapter.

**Note:** When modifying previous statements by reissuing them, the parmlib statements have a cumulative effect; that is, any one parmlib member might not reflect the current configuration. If you use the CANCEL command to terminate the APPC address space, you must re-specify each parmlib member in its former order to reconstruct the previous configuration.

### Default Values

When APPCPMxx statements omit optional parameters, the system usually supplies default values. When the statement is issued for the first time, the system always supplies default values for omitted optional parameters. When the statement modifies a previous statement, the system overrides all previously specified values with default values except for the SCHED, TPDATA, TPLEVEL, USERVAR and ALTLU parameters of the LUADD statement.

**Note:** To guarantee that intended values are not overridden by default values, re-specify all parameters on modifying statements.

When an LU is added and certain parameters are omitted, those parameters receive default values. For example, the following LUADD statement supplies only the name of the new LU.

```
LUADD  
  ACBNAME(MVSLU04)
```



The remaining keyword parameters receive default values so the LUADD statement is equivalent to the following:

```
LUADD
  ACBNAME(MVSLU04)
  SCHED(ASCH)
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)
```

IBM does not supply a default APPCPMxx parmlib member; however, a sample member that you can modify is in the APPCPMXX member of SYS1.SAMPLIB.

## Planning Specific Values

---

The values in APPCPMxx parmlib members control the APPC/MVS communication configuration. This section gives some guidance on the following tasks involving installing and customizing the communication aspects of APPC/MVS.

- Adding a local LU
- Modifying a local LU
- Deleting a local LU
- Specifying the VSAM KSDS for side information
- Modifying the name of the VSAM KSDS for side information.

For specific coding details for the APPCPMxx parmlib member, see [\*z/OS MVS Initialization and Tuning Reference\*](#).

## Adding a Local LU — LUADD Statement

---

The LUADD statement defines a local APPC/MVS LU that is to be added to the APPC configuration. Each LU on MVS must be defined with an LUADD statement that specifies a name for the LU, through the ACBNAME parameter. Other LUADD statement parameters are:

### **SCHED(name) or NOSCHED**

Indicates whether the LU is associated with a transaction scheduler

### **BASE**

Specifies whether the LU is a base LU

### **PSTIMER**

Indicates the length of time that the sessions persist

### **TPDATA(name)**

Identifies the TP profile file associated with the LU

### **TPLEVEL(level)**

Identifies the level of TP for which the LU searches

### **ALTLU and USERVAR**

Pass optional, installation-supplied data to an alternate transaction scheduler

### **GRNAME(genericname)**

Specifies a VTAM generic resource name to be associated with the LU

### **NQN or NONQN**

Specifies whether the LU is enabled to use a network-qualified partner LU name when first allocating an outbound conversation.

When an installation uses the APPC/MVS transaction scheduler exclusively, only one LU is required. If other transaction schedulers are used, each scheduler requires a separate LU. An installation might also choose to define additional LUs for use with APPC/MVS servers, or to isolate TPs for security or testing.



## Example of Adding LUs

The following example shows four LUADD statements in a parmlib member named APPCPM1A. The first LU is similar to the default LU provided in the sample APPCPMXX member in SYS1.SAMPLIB. It is a member of the VTAM generic resource group MVSLU1, and is enabled for network-qualified names support. The second LU is one in a generic resource group of test LUs that access group and system-level profiles from a special test TP profile file. The third LU accesses user, group, and system-level profiles from the TP profile file SYS1.XYZTP. Following an interruption, sessions in this LU would persist for 3600 seconds (1 hour). If service is not restored within 3600 seconds the LU becomes unavailable. The USERVAR and ALTLU keywords are used to pass data to scheduler XYZ. This XYZ scheduler is a scheduler other than the APPC/MVS transaction scheduler (ASCH). The fourth LU is a NOSCHED LU from which APPC/MVS servers will receive inbound conversations. This LU accesses only the database token from TP profile file SYS1.APPCTP.

```
LUADD
  ACBNAME(Z098AP01)
  SCHED(ASCH)
  GRNAME(MVSLU1)
  NQN
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)

LUADD
  ACBNAME(Z098AP02)
  SCHED(ASCH)
  GRNAME(MVSTEST)
  NQN
  TPDATA(SYS1.APPCTEST)
  TPLEVEL(GROUP)

LUADD
  ACBNAME(Z096AP02)
  SCHED(XYZ)
  BASE
  NQN
  TPDATA(SYS1.XYZTP)
  TPLEVEL(USER)
  PSTIMER(3600)
  USERVAR(scheduler-supplied value)
  ALTLU(scheduler-supplied value)

LUADD
  ACBNAME(Z096AP03)
  NOSCHED
  GRNAME(SERVLU)
  NQN
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)
```

Figure 71. APPCPM1A

To activate the LUs, issue the START command if APPC was not previously started; otherwise issue the SET command. Examples of each command follow.

```
START APPC,SUB=MSTR,APPC=1A

SET APPC=1A
```

## Modifying a Local LU – LUADD Statement

You can modify an LU by overriding a previous LUADD statement with another LUADD statement that names the existing LU and changes the definition. The LU definitions you can modify are:

### BASE

Whether the LU is base

**TPDATA(name)**

TP profile file associated with the LU

**TPLEVEL(level)**

Level of TP for which the LU searches

**PSTIMER**

Length of time that the sessions persist

To change other parameter values that define an existing LU, you must delete the LU with an LUDEL statement and re-identify the LU with a new LUADD with different values for those parameters. The two statements, the LUDEL and the new LUADD, cannot be in the same parmlib member; and, if in two parmlib members, the two parmlib members cannot be specified in the same SET command.

Modifying previous LUADD statements is useful when an installation needs to specify a different configuration for the same LU on a regular basis. For example, the installation might need one LU definition for a particular kind of work during first shift, and another definition for the same LU for another kind of work during second shift. The installation can keep each definition in a separate parmlib member.

## Examples of Modifying an LU

In the following example, the first LUADD statement in parmlib member APPCPM2A defines LU Z098AP01, and the second LUADD statement in APPCPM2M changes the TP profile file for the LU.

```
LUADD
ACBNAME(Z098AP01)
SCHED(ASCH)
GRNAME(MVSLU1)
NQN
TPDATA(SYS1.APPCTP)
TPLEVEL(SYSTEM)
```

*Figure 72. APPCPM2A*

```
LUADD
ACBNAME(Z098AP01)
SCHED(ASCH)
GRNAME(MVSLU1)
NQN
TPDATA(SYS1.NEWTP)
TPLEVEL(SYSTEM)
```

*Figure 73. APPCPM2M*

Assuming that parmlib member APPCPM2A is already active, you can modify the LU by issuing the SET command as follows:

```
SET APPC=2M
```

You can also modify the level of an LU through a parmlib member that contains the original LUADD statement with a different TPLEVEL parameter. The following LUADD modifies the level of the LUADD in parmlib member APPCPM2A from SYSTEM to USER.

```

LUADD
ACBNAME(Z098AP01)
SCHED(ASCH)
GRNAME(MVSLU1)
NQN
TPDATA(SYS1.APPCTP)
TPLEVEL(USER)

```

Figure 74. APPCPM3M

Regardless of whether parmlib member APPCPM2A or APPCPM2M is active, to make the change, issue the SET command as follows:

```
SET APPC=3M
```

If after several modifications, you are unsure of what your configuration looks like, issue the following DISPLAY APPC command.

```
DISPLAY APPC,LU,ALL
```

Assuming you created and modified LUs using the previous examples and sessions were bound, you would see the following:

```

ATB121I 15.55.45 APPC DISPLAY      FRAME 1   F   E   SYS=SY1
ACTIVE LU'S   OUTBOUND LU'S   PENDING LU'S   TERMINATING LU'S
00006        00000        00000        00000
SIDEINFO=SYS1.APPCSI
LLUN=Z098AP01  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001    TPLEVEL=USER   SYNCPT=NO
GRNAME=MVSLU1  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
LLUN=Z098AP02  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00000    TPLEVEL=GROUP  SYNCPT=NO
GRNAME=MVSTEST RMNAME=*NONE*
TPDATA=SYS1.APPCTEST
LLUN=Z096AP02  SCHED=XYZ       BASE=YES       NQN=YES
STATUS=ACTIVE  PARTNERS=00001    TPLEVEL=USER   SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.XYZTP
PLUN=USIBMZ0.MVSLU4
LLUN=Z096AP03  SCHED=*NONE*     BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00003    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=SERVLU  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU
PLUN=USIBMZ0.MVSLU
PLUN=USIBMZ0.MVSLU4
LLUN=Z098AP04  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00000    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
LLUN=Z096AP04  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU   RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1

```

Figure 75. DISPLAY command output

## Deleting a Local LU – LUDEL Statement

To delete an APPC/MVS local LU from an APPC configuration, use the LUDEL statement, specifying the LU name through the ACBNAME parameter.

Other LUDEL statement parameters are:

## PERSIST or NOPERSIST

Indicates whether APPC/MVS should deactivate persistent sessions between the LU and its partners.

When an LUDEL statement is processed, new allocation requests to the named LU are rejected; however, all existing conversations are allowed to continue until completed. The LU is removed only after all existing conversations have ended. At that point, you may use an LUADD to bring up the LU again.

Before deleting an LU, be aware of the network implications of removing the LU. Do peer systems depend on the LU as a partner LU? Is the MVS LU name defined in their communications setup and should they be notified of the deletion? What is the impact on network communications when the LU is deleted? See [“Stopping APPC/MVS Work” on page 188](#) for more information about the effects of stopping APPC/MVS LUs.

## Examples of Deleting an LU

To delete an LU (such as, a test LU that is no longer needed) code a parmlib member that contains the LUDEL statement followed by the name of the LU to be deleted.

```
LUDEL
ACBNAME(Z098AP02)
```

*Figure 76. APPCPM1D*

To activate the parmlib member containing the LUDEL, issue the SET command as follows:

```
SET APPC=1D
```

In the following example, the LU Z098AP02 has been added to the APPC configuration with a PSTIMER value. The LUDEL statement in parmlib member APPCPM2D deletes LU Z098AP02 from the APPC configuration, but does not delete any persistent sessions.

```
LUDEL
ACBNAME(Z098AP02) PERSIST
```

*Figure 77. APPCPM2D*

To activate the parmlib member containing the LUDEL, issue the SET command as follows:

```
SET APPC=2D
```

When this LU is now added back into the APPC configuration, all the sessions between this LU and all of its partners will not be disrupted. See [“Optimize LU-to-LU Sessions” on page 219](#) for more information about persistent sessions.

## Specifying a VSAM KSDS for Side Information – SIDEINFO Statement

The SIDEINFO statement specifies the VSAM key sequenced data set (KSDS) in which side information entries are kept. The SIDEINFO statement contains the name of the side information file—DATASET(*name*).

If you do not specify the SIDEINFO statement, no default side information file name is activated.

Unlike TP profile files, only one side information file is allowed per z/OS system. The VSAM file must have been previously defined and be cataloged in either a user catalog or the master catalog.

You can name a different VSAM KSDS as the system side information file by overriding the DATASET parameter with another DATASET parameter.

For a sample VSAM file definition, see SYS1.SAMPLIB member ATBSIVSM.

## Examples Using APPCPMxx Parmlib Members

Because of the cumulative way the APPCPMxx parmliib members function, you might consider creating a separate member for:

- Initial APPC setup
- Each anticipated modification
- Deletion of each LU and each unique session.

### Initial APPC Setup

In the following example, an initial APPC setup might consist of three LUs, one unique session definition, and one side information definition.

```
LUADD
  ACBNAME(Z098AP01)
  SCHED(ASCH)
  GRNAME(MVSLU1)
  NQN
  TPDATA(SYS1.APPCTP)
  TPLEVEL(SYSTEM)

LUADD
  ACBNAME(Z098AP02)
  SCHED(ASCH)
  GRNAME(MVSTEST)
  NQN
  TPDATA(SYS1.APPCTEST)
  TPLEVEL(GROUP)

LUADD
  ACBNAME(Z096AP02)
  SCHED(XYZ)
  BASE
  NQN
  TPDATA(SYS1.XYZTP)
  TPLEVEL(USER)
  PSTIMER(3600)
  USERVAR(scheduler-supplied value)
  ALTLU(scheduler-supplied value)

SIDEINFO
  DATASET(SYS1.APPCSI)
```

*Figure 78. APPCPM1A*

### Anticipated Modifications

When an installation needs to change its LU configuration regularly, parmliib members can be pre-coded with the changes. For example, if an installation needs two system-level production LUs during third shift, LU Z096AP02 can be changed every evening from a user-level LU to a system-level production LU. The installation can code two parmliib members— one to change the LU to system level and one to change it back to user level.

Parmliib member APPCPM3S changes LU Z096AP02 to a system-level LU and member APPCPM1S changes it back to a user-level LU. When the installation wants LU Z096AP02 to be system level for third shift, the operator issues SET APPC=3S. When the installation wants LU Z096AP02 to be user level again, the operator issues SET APPC=1S.

```

LUADD
  ACBNAME(Z096AP02)
  SCHED(XYZ)
  BASE
  NQN
  TPDATA(SYS1.XYZTP)
  TPLEVEL(SYSTEM)
  PSTIMER(3600)
  USERVAR(scheduler-supplied value)
  ALTLU(scheduler-supplied value)

```

*Figure 79. APPCPM3S*

```

LUADD
  ACBNAME(Z096AP02)
  SCHED(XYZ)
  BASE
  NQN
  TPDATA(SYS1.XYZTP)
  TPLEVEL(USER)
  PSTIMER(3600)
  USERVAR(scheduler-supplied value)
  ALTLU(scheduler-supplied value)

```

*Figure 80. APPCPM1S*

Note that during third shift, there is no one parmlib member that reflects the configuration. Rather, the configuration is a combination of two parmlib members, APPCPM1A and APPCPM3S.

## Deletions

If an installation needs to delete an LU from the configuration or delete a previously specified unique session definition, it can have available pre-coded parmlib members, each of which contains a delete.

The following examples are parmlib members that delete each of the LUs and the one unique session in APPCPM1A.

```

LUDEL
  ACBNAME(Z098AP01)

```

*Figure 81. APPCPM1D*

```

LUDEL
  ACBNAME(Z098AP02)

```

*Figure 82. APPCPM2D*

```

LUDEL
  ACBNAME(Z096AP02)

```

*Figure 83. APPCPM3D*

When the installation needs to delete an LU or the unique session, an operator can issue a SET APPC command followed by the identifier of the parmlib member containing the delete.

## Tracking Changes in the Configuration

There are two ways to keep track of parmlib changes:

- Keep a hardcopy log of every APPCPMxx member that was activated by using the LIST option on the START APPC and SET APPC commands.
- View the current communication configuration by issuing the DISPLAY APPC LU,ALL command.

### Keeping a Hardcopy Log

You can define on the HARDCOPY statement of a CONSOLxx parmlib member, a hardcopy log that provides a permanent record of APPC parmlib activity. For information about defining the hardcopy log, see *z/OS MVS Planning: Operations*.

To list the contents of each activated parmlib member to the operator console and to the hardcopy log, include the LIST option on the START and SET commands. For example, when starting APPC using parmlib member APPCPM1A, issue the START command as follows:

```
START APPC,SUB=MSTR,APPC=(1A,L)
```

When changing the configuration with parmlib member APPCPM3S and APPCPM2D, issue the SET command with the LIST option as follows:

```
SET APPC=(3S,2D,L)
```

This displays the contents of both APPCPM3S and APPCPM2D on the console screen and stores the information in the hardcopy log.

```
ASB038I APPCPM3S : LUADD
ASB038I APPCPM3S : ACBNAME(Z096AP02)
ASB038I APPCPM3S : SCHED(XYZ)
ASB038I APPCPM3S : BASE
ASB038I APPCPM3S : NQN
ASB038I APPCPM3S : TPDATA(SYS1.XYZTP)
ASB038I APPCPM3S : TPLEVEL(SYSTEM)
ASB038I APPCPM3S : PSTIMER(3600)
ASB038I APPCPM3S : USERVAR(scheduler-supplied value)
ASB038I APPCPM3S : ALTLU(scheduler-supplied value)

ASB038I APPCPM2D : LUDEL
ASB038I APPCPM2D : ACBNAME(Z098AP02)
```

*Figure 84. SET command LIST option output*

### Viewing the Current Configuration

A way to get a "snapshot" of the current configuration is with the DISPLAY command. To view the LU configuration, issue the DISPLAY command as follows:

```
DISPLAY APPC,LU,ALL
```

```

ATB121I 15.48.39 APPC DISPLAY      FRAME 1      F      E  SYS=SY2
ACTIVE LU'S      OUTBOUND LU'S    PENDING LU'S  TERMINATING LU'S
00003           00000             00000           00000
SIDEINFO=SYS1.APPCSI
LLUN=Z098AP01    SCHED=ASCH        BASE=NO        NQN=YES
STATUS=ACTIVE    PARTNERS=00000    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU1    RMNAME=*NONE*
TPDATA=SYS1.APPCTP
LLUN=Z098AP02    SCHED=ASCH        BASE=NO        NQN=YES
STATUS=ACTIVE    PARTNERS=00001    TPLEVEL=GROUP  SYNCPT=NO
GRNAME=MVSTEST   RMNAME=*NONE*
TPDATA=SYS1.APPCTEST
PLUN=USIBMY0.MVSLU1
LLUN=Z098AP04    SCHED=ASCH        BASE=NO        NQN=YES
STATUS=ACTIVE    PARTNERS=00002    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*    RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
PLUN=USIBMY0.Z096AP02

```

Figure 85. DISPLAY command output



---

## Part 4. Security management



---

## Chapter 10. Setting up Network Security

APPC/MVS provides a number of security mechanisms that you can use to maintain network security in a cooperative processing environment. This chapter describes those mechanisms and ways to select and implement them based on the needs of your applications and installation. The audience for this chapter is system programmers and security administrators.

References:

[\*z/OS Security Server RACF System Programmer's Guide\*](#)  
[\*z/OS Security Server RACF Security Administrator's Guide\*](#)  
[\*z/OS Security Server RACF Command Language Reference\*](#)  
[\*z/OS Security Server RACF Messages and Codes\*](#)  
[\*z/OS Communications Server: SNA Resource Definition Reference\*](#)  
[\*z/OS Communications Server: SNA Operation\*](#)  
[\*z/OS MVS System Messages, Vol 3 \(ASB-BPX\)\*](#)

---

### APPC/MVS Security Requirements

This chapter assumes that you are using Resource Access Control Facility (RACF) as your installation's security product. You could use an equivalent security product instead of RACF. You must activate the security product when you IPL your system.

**Note:** All TP profile names that you want to protect with RACF must be in uppercase. RACF does not process lowercase TP profile names.

Most of the APPC/MVS security support is optional or automatic. The only security requirement that must be met before using APPC/MVS is ensuring that the APPC and ASCH started procedures can access the resources that they need.

### Giving the APPC and ASCH Started Procedures Access to Resources

Before you can start APPC and the APPC/MVS transaction scheduler (ASCH), the APPC and ASCH started procedures must have read-only access to the parmlib concatenation.

Your installation probably limits access to the parmlib concatenation to only authorized users. If so, you must refer to security product documentation for protecting and authorizing started procedures. If your installation uses RACF, see [\*z/OS Security Server RACF Security Administrator's Guide\*](#) for information about giving read access to the APPC and ASCH started procedures.

Although your installation might also limit access to TP profile and side information data sets, you do not have to explicitly authorize the APPC and ASCH procedures to access those data sets. Because the APPC and ASCH started procedures reside in an authorized library, no additional security authorization is required for them to access TP profiles and side information.

The rest of this chapter describes the APPC/MVS security mechanisms, starting with a brief review of cooperative processing and its security implications.

---

### Why Security for APPC?

Cooperative processing allows application programs to establish communications with partner programs on other systems, and to share work, data, and services between systems and across networks. This ability to access other programs and all the resources at their disposal poses special security considerations for installations that use cooperative processing.

APPC/MVS is a cooperative processing interface on MVS/ESA. With APPC/MVS, transaction programs (TPs) on MVS can initiate (allocate) conversations with partner programs on systems throughout an SNA network. The partner programs can likewise allocate conversations with TPs on MVS. In an unprotected

network, all a TP has to know to start a conversation is the name of an inbound TP and the logical unit (LU) on which the inbound TP is located. Unless certain precautions are taken, it is possible for unauthorized conversations to take place. To protect your z/OS system from unauthorized conversation requests, you might want to take some of the following steps:

- Limit the logical units from which conversation requests can enter your system
- Ensure that inbound requests for conversations with your system contain security information such as a user ID and password
- Limit, by user ID, those users who can request a particular TP on your system
- Limit the administrators who can define TPs to APPC/MVS
- Ensure that TPs on MVS run in the appropriate security environment, one that represents the requester of the MVS TP.
- Minimize the flow of passwords across the network.

This chapter discusses these and other security mechanisms for cooperative processing and describes how you can implement them using APPC/MVS and RACF.

## An APPC Application Example

Figure 86 on page 134 depicts a typical APPC application. The application consists of two transaction programs, TPA and TPB, which hold a conversation between different systems in an SNA network. Their conversation occurs across a session between their logical units, LU01 and LU02, which represent points of entry into the network from their systems. This example is repeated and adapted throughout the chapter to illustrate the different mechanisms for LU and conversation security.

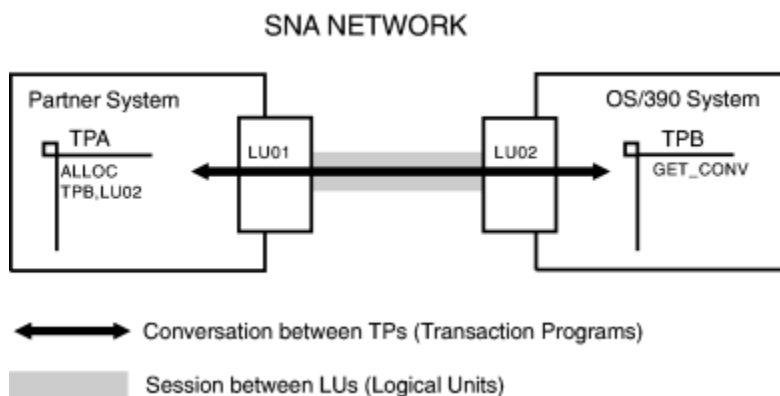


Figure 86. Sample APPC/MVS Conversation

In the example, TPA is the outbound TP, which starts (allocates) the conversation. The conversation is considered outbound from TPA. TPB is the inbound TP; it receives and processes the conversation request from TPA. From TPB's point of view, the conversation is inbound.

Most of the APPC/MVS security mechanisms protect inbound TPs and LUs on MVS, such as TPB and LU02, from unauthorized inbound requests.

## Planning for APPC Security

Any plan for APPC security must begin with the APPC applications that you have installed or plan to install. The applications themselves, and the sensitivity of the data and resources that they use, will dictate the level of security that you need. If the TPs do not involve sensitive data, or if they communicate between two equally trusted systems, you might not need to use any of the APPC security mechanisms. If, however, a TP on MVS uses sensitive data or functions, or processes requests from partner TPs at remote workstations, you might want to activate some or all of the security mechanisms to protect the application.

The APPC applications themselves can provide a basis for security by passing security information on the allocate request that starts the conversation. The system where the inbound TP is located can verify the

inbound security information (a combination of user ID, password, and security profile name), can permit or deny the conversation request accordingly and, in the case of MVS, can schedule the inbound TP to run in a security environment based on the inbound security information.

## Determining the Application's Security Type

The first step in protecting an application is to determine what security information, if any, the inbound TP requires. APPC applications that you write or install will provide one of three types of security, as specified on the Allocate call:

### NONE

The outbound TP passes no security information. The conversation cannot use conversation security mechanisms.

### SAME

The outbound TP indicates that the inbound TP should have the same security as the outbound TP. APPC provides the following security information, if any is available, from the outbound TP to the inbound:

- A user ID
- A security profile name, which APPC/MVS treats as a group ID
- An *already verified* (AV) indicator.

This security information is obtained from a number of different sources, depending on the current execution environment and the input parameters specified. If the user is authorized, uses an MVS-specific Allocate service (for example, ATBALC5), and specifies a valid User-Token parameter, APPC will use this to obtain a user ID and, if available, a profile name. If this is not specified, APPC will send the user ID associated with the current application work context, if this is available. Otherwise, APPC will send the user ID and, if available, a profile name that is associated with the current executing task, or if unavailable, from the current address space.

If no security information is available for the outbound TP, or the inbound system does not support *already verified*, no security information is passed and the security type is treated as NONE.

(If an allocate request is made from MVS using CMALLC, the CPI Communications Allocate call, the security type is always SAME.)

### PGM

The outbound TP specifies a user ID, password and optional security profile name, which RACF treats as a group ID. The outbound TP passes these to the inbound system for verification.

An alternative to specifying both a user ID and a password is to specify a user ID only, provided that the TP specifying the user ID has the necessary surrogate authority to the specified user ID. Requests that use this technique will send the already verified (AV) indicator to the inbound TP.

For a general description of surrogate user authorization, see [z/OS Security Server RACF Security Administrator's Guide](#). The specific steps needed to set up surrogate authorization for APPC/MVS are as follows:

1. Identify the user ID associated with the address space in which the TP is issuing the Allocate request. For batch jobs and TSO/E sessions, a user ID is already associated with the address space. For started tasks, the security administrator needs to associate a user ID with the address space of the started task, using the STARTED class in RACF, the RACF started procedure table (SPT), or both (recommended).
2. For each user ID that needs to use PGM security without a password, define a resource profile in the SURROGAT class as follows:

```
RDEFINE SURROGAT ATBALLC.userid UACC(NONE)
```

If generic profile checking for the SURROGAT class has been activated by the RACF security administrator, you can also create generic profiles to allow APPC/MVS surrogate authorization for multiple users. If SURROGAT profiles named ATBALLC.\* and ATBALLC.FRED both existed,

ATBALLC.FRED would control APPC/MVS surrogate authorization for FRED, and ATBALLC.\* would control APPC/MVS surrogate authorization for all other users. For more information on activating generic profile checking in RACF, see *z/OS Security Server RACF Security Administrator's Guide*.

3. Give the APPC TP that needs surrogate authorization READ access to the ATBALLC.userid profile created in the previous step:

```
PERMIT ATBALLC.userid CLASS(SURROGAT) ID(userid-of-the-APPC-TP) ACCESS(READ)
```

where

**userid**

is the user ID that the TP specifies on the Allocate call without a password.

**userid-of-the-APPC-TP**

is the user ID associated with the address space in which the TP is issuing the Allocate request.

4. For this APPC/MVS support to function, you must activate the SURROGAT class. In addition, APPC/MVS requires that the SURROGAT class be RACLISTed:

```
SETROPTS CLASSACT(SURROGAT) RACLIST(SURROGAT)
```

At this point, the user ID associated with the TP has authorization to specify a user ID other than its own without specifying a password.

When the inbound TP resides on MVS, each field of security information (user ID, password, and optional security profile) on an allocate request must not exceed 8 non-blank characters in length. If any of the characters are lowercase, APPC/MVS changes them to uppercase before user verification.

Figure 87 on page 136 shows an example of providing security information on an allocate request. In the figure, TPA specifies a security type of PGM and passes a user ID and password. TPB's system uses RACF security mechanisms to verify that information, verify access to TPB, and set up TPB's security environment. If TPA specified a security type of SAME, APPC itself would extract and pass any available security information on the allocate request.

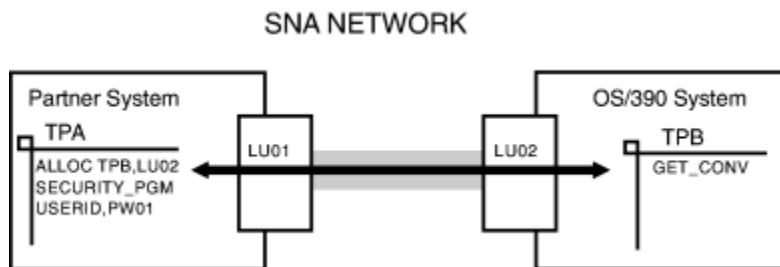


Figure 87. Passing Security\_PGM Information on an Allocate Request

If you have an APPC application with an inbound TP on MVS, once you know what security information the outbound TP is passing on its allocate request, you can decide how you want to verify and use that information. In other words, you can decide what RACF security mechanisms to use to protect the inbound TP and its LU.

The security mechanisms are divided into two categories:

- LU security to protect APPC/MVS logical units
- Conversation security to protect APPC/MVS transaction programs.

## LU Security Mechanisms

APPC/MVS provides the following security mechanisms to protect logical units assigned to APPC/MVS:

- Specifying VTAM security keywords
- Allowing LU-to-LU security verification
- Controlling the use of VTAM ACBs.

For more information about these LU security mechanisms, see [“LU Security: Protecting APPC/MVS Logical Units”](#) on page 137.

## Conversation Security Mechanisms

In addition to LU security mechanisms, for APPC applications whose security type is SAME or PGM (for which a user ID is passed on the allocate request), APPC/MVS provides the following conversation security mechanisms as well:

- Establishing a TP security environment on MVS
- Controlling user access to LUs
- Controlling user access from LUs
- Controlling user access to TP profiles and side information, and tailoring TP work attributes
- Controlling the ability to collect API trace data for conversations
- Persistent verification (PV)

For more information about these conversation security mechanisms, see [“Conversation Security: Protecting APPC/MVS TPs”](#) on page 144.

Remember, the APPC/MVS security mechanisms primarily protect MVS and the TPs that run on it. Other operating systems that support APPC, such as Microsoft Windows, Sun Solaris, AIX, OS/400, OS/2, and VM, generally offer their own security mechanisms to protect TPs at their end of a conversation. If your cooperative processing applications include TPs on systems other than MVS, consult that system's documentation for information on protecting those TPs.

## LU Security: Protecting APPC/MVS Logical Units

---

Before APPC applications can communicate, you must define logical units (LUs) to VTAM. You define the LUs to VTAM by coding VTAM APPL statements, as described in Chapter 8, “Planning Sessions,” on page 95. The LUs represent nodes, or points of entry into the network; each transaction program is associated with an LU.

As the point of entry for APPC communications into your system, an LU might require special protection. There are several steps you can take to protect APPC/MVS LUs from unauthorized access:

- Specifying security keywords on VTAM APPL statements

You can include security information on the VERIFY and SECACPT keywords of the APPL statement to make VTAM verify LU-to-LU session requests and accept default levels of conversation security between LUs

- Allowing LU-to-LU security verification with APPCLU profiles

For each LU on MVS, you can specify the partner LUs with which it can hold sessions, through a session key that VTAM verifies. And for each pair of LUs, you can specify the levels of security that you will allow on conversations that cross their sessions.

- Controlling the use of VTAM ACBs

You can ensure that an LU is defined to VTAM from the APPC address space only, using RACF VTAMAPPL profiles.

## Coding Security Keywords on the VTAM APPL Statement

When you code a VTAM APPL statement to define an LU to VTAM, you can specify:

- The level of TP security that VTAM will accept in conversation requests for TPs at the LU (SECACPT keyword)
- Whether VTAM is to verify the identity of partner LUs (VERIFY keyword).

## Specifying the Level of Conversation Security for VTAM

As described in [Chapter 8, “Planning Sessions,”](#) on page 95, conversation security is a factor in planning LUs and defining them to VTAM. On the VTAM APPL statement that defines an LU to VTAM, you must specify the greatest level of security to be allowed on inbound conversation requests for TPs at the LU. You do this by specifying one of the following values on the APPL statement's SECACPT keyword:

### Value

**Means VTAM will accept:**

#### NONE

Requests that contain no security information (the default)

#### CONV

Requests with security information specified

#### ALREADYV

Requests with security information specified, and requests with an indication that security information is already verified (includes CONV). Use only between trusted LUs.

#### PERSISTV

Requests with security information specified, and requests with an indication of persistent verification. For more information about persistent verification, see [“Using Persistent Verification \(PV\)”](#) on page 157.

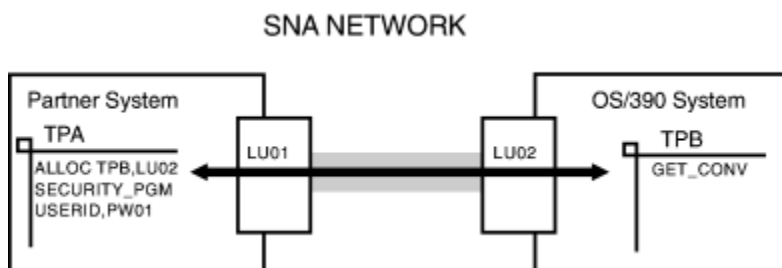
#### AVPV

Requests with security information specified, and requests with indications of already verified or persistent verification.

The value you specify in the SECACPT keyword must be appropriate for the TPs that are to use the LU. SECACPT must allow the type of security information that the TPs expect to receive. For example, the following SECACPT value would be appropriate for LU02 and TPB as shown in [Figure 88](#) on page 138:

```
APPL ACBNAME=LU02... SECACPT=CONV
```

The above statement tells VTAM to accept conversation requests for LU02 that have security information (user ID and password) such as TPA specifies.



*Figure 88. Sending Security Information through VTAM*

Suppose the APPL definition for LU02 specified SECACPT=NONE instead of CONV, and TPA issues the same Allocate call, as shown in [Figure 88](#) on page 138. The security information for the outbound TP is greater than the SECACPT value of the partner LU. In such cases, the system downgrades or removes security information from the outbound Allocate request, so that the request matches the minimum security requirements for the partner LU. The results might not be what you expected for this conversation.

The SECACPT value that you specify on the VTAM APPL statement provides the default level of acceptable conversation security. You can override that level using the RACF APPCLU profiles, as described in [“Defining Conversation Security Levels that Sessions Allow”](#) on page 142.



## Requesting that VTAM Verify Partner LUs

When your installation codes a VTAM APPL statement to define an LU to VTAM, you must specify whether you want VTAM to verify the identity of partner LUs that attempt to establish sessions with the LU. To do so, set the APPL statement's VERIFY keyword to one of the following values:

### Value

#### Meaning

### NONE

VTAM should not verify partner LUs (the default).

### OPTIONAL

VTAM should verify those partner LUs that have defined LU-to-LU passwords (session keys), as described in *z/OS Communications Server: SNA Resource Definition Reference*.

### REQUIRED

VTAM should verify every partner LU.

If you request verification, VTAM verifies partner LUs by means of an associated session key. You can use the RACF APPCLU class to specify the session key for each LU, as described in [“Defining LU-to-LU Session Keys”](#) on page 142.

## Defining LU-to-LU Access Authority with RACF APPCLU Profiles

After you decide which LUs may hold conversations, you can create RACF profiles to define more security characteristics for LUs and for conversations between the LUs. To do so, create RACF profiles in the APPCLU class. These profiles differ, depending on whether the LU is a member of a VTAM generic resource group:

- For information about protecting LUs that are not members of a generic resource group, see [“Defining LU-to-LU Access Authority for a Specific LU”](#) on page 139.
- For information about protecting LUs that are members of a generic resource group, see [“Defining LU-to-LU Access Authority for LUs in a VTAM Generic Resource Group”](#) on page 140.

You may also define session keys and conversation security levels for LU-to-LU security; see:

- [“Defining LU-to-LU Session Keys”](#) on page 142, or
- [“Defining Conversation Security Levels that Sessions Allow”](#) on page 142.

## Defining LU-to-LU Access Authority for a Specific LU

To define LU-to-LU access authority for a specific LU and one or more of its partners, use the RDEFINE command for the APPCLU class. The syntax for RDEFINE depends on whether the LU is enabled to support network-qualified names (that is, whether the NQN parameter is specified on the LUADD statement in parmlib member APPCPMxx):

- If the LU is enabled to support network-qualified names (NQN is specified on the LUADD statement), the RDEFINE syntax is:

```
RDEFINE APPCLU 1network-id.local-lu-name.pnetwork-id.partner-lu-name
UACC(NONE)
```

If you are enabling an existing LU to use network-qualified names, you must complete these APPCLU definitions before issuing the SET command for the parmlib member that contains the LUADD statement with the NQN parameter. See [“Using Network-Qualified Names Support”](#) on page 96 for a list of the steps required to enable network-qualified names support.

- If the LU is **not** enabled to support network-qualified names (NONQN is specified on, or used as the default for, the LUADD statement), the RDEFINE syntax is:

```
RDEFINE APPCLU 1network-id.local-lu-name.partner-lu-name
UACC(NONE)
```

In the RDEFINE syntax, variables are defined as follows:

**lnetwork-id or pnetwork-id**

Is the network ID for the network on which the local LU or partner LU resides. This value is 1 through 8 characters, and matches the value coded for the VTAM start option NETID. RACF requires this value to be in discrete form (that is, the value cannot contain any wildcard characters).

**local-lu-name**

Is the network name of the local LU. This value is 1 through 8 characters, and matches the application name coded on the APPL statement. RACF requires this value to be in discrete form (that is, the value cannot contain any wildcard characters).

**partner-lu-name**

Is the network name of the partner LU; that is, the 1- through 8-byte network-LU-name portion of their network-qualified names. RACF accepts this value in generic form (that is, the value can contain wildcard characters).

On the partner LU's system, you need a corresponding definition of the two LUs. If RACF is installed on the partner system, define a corresponding APPCLU profile there, with the proper network id and with the LU names in reverse order. For example, if LU01 and LU02 are both enabled for network-qualified names, and reside on network USIBMZ0, you would need to specify the following commands on the LUs' respective systems:

```
RDEFINE APPCLU USIBMZ0.LU01.USIBMZ0.LU02 UACC(NONE)
RDEFINE APPCLU USIBMZ0.LU02.USIBMZ0.LU01 UACC(NONE)
```

If LU01 were on network USIBMZ0, and LU02 were on network USIBMZ3, you would specify, on their respective systems:

```
RDEFINE APPCLU USIBMZ0.LU01.USIBMZ3.LU02 UACC(NONE)
RDEFINE APPCLU USIBMZ3.LU02.USIBMZ0.LU01 UACC(NONE)
```

If LU01 and LU02 are not enabled for network-qualified names, and reside on network USIBMZ0, you would need to specify the following commands on the LUs' respective systems:

```
RDEFINE APPCLU USIBMZ0.LU01.LU02 UACC(NONE)
RDEFINE APPCLU USIBMZ0.LU02.LU01 UACC(NONE)
```

Next, you can define session keys for the corresponding LUs as described in [“Defining LU-to-LU Session Keys”](#) on page 142.

If the partner system is OS/400, see *AS/400 APPC Programmer's Guide*.

## Defining LU-to-LU Access Authority for LUs in a VTAM Generic Resource Group

If the LUs are members of a VTAM generic resource group, you have a choice about defining APPCLU profiles for them:

- You can define many APPCLU profiles, one for each LU that is a member of a VTAM generic resource group. These profiles have names that include `local-lu-name`, as described in [“Defining LU-to-LU Access Authority for a Specific LU”](#) on page 139.
- You can define one APPCLU profile for each VTAM generic resource group, as described in this section. These profiles have names that include `generic-resource-group-name`, as shown in the examples to follow.

**Migration Note:** If you originally have APPCLU profiles for specific LUs (with `local-lu-name` in the profile names), and you later add APPCLU profiles for LUs that are in VTAM generic resource groups (with `generic-resource-group-name` in the profile names), you must delete the APPCLU profiles for specific LUs. If both kinds of APPCLU profiles exist, only the profiles for specific LUs are used.

To determine which specific LUs are in a VTAM generic resource group, issue the following command:

```
DISPLAY NET,ID=generic-resource-group-name
```

For example, if the VTAM generic resource group name is GENERNAM, issue:

```
d net,id=genernam
```

If the output is:

```
JOB      2  IST097I DISPLAY ACCEPTED
JOB      2  IST075I NAME = GENERNAM, TYPE = GENERIC RESOURCE
IST1359I MEMBER NAME          OWNING CP  SELECTABLE  APPC
IST1360I NETA.APPCAP06         SSCP2A      YES         YES
IST1360I NETA.APPCAP05         SSCP1A      YES         YES
IST1393I GENERIC RESOURCE NAME RESOLUTION EXIT IS ISTEXCGR
IST314I END
```

To cause VTAM to refresh updated RACF profiles for generic name GENERNAM, issue MODIFY PROFILES,ID=APPCAP05 on the SSCP1A host where APPCAP05 resides, and MODIFY PROFILES,ID=APPCAP06 on the SSCP2A host where APPCAP06 resides.

A further consideration relates to network-qualified names:

- If the LUs in the group are enabled to support network-qualified names (NQN is specified on each LUADD statement), the RDEFINE syntax is:

```
RDEFINE APPCLU lnetwork-id.generic-resource-group-name.pnetwork-id.partner-lu-name
UACC(NONE)
```

- If the LUs are **not** enabled to support network-qualified names (NONQN is specified on, or used as the default for, each LUADD statement), the RDEFINE syntax is:

```
RDEFINE APPCLU lnetwork-id.generic-resource-group-name.partner-lu-name UACC(NONE)
```

In the RDEFINE syntax, variables are defined as follows:

#### **lnetwork-id or pnetwork-id**

Is the network ID for the network on which the local LU or partner LU resides. This value is 1 through 8 characters, and matches the value coded for the VTAM start option NETID. RACF requires this value to be in discrete form (that is, the value cannot contain any wildcard characters).

#### **generic-resource-group-name**

Is the generic resource name associated with the LUs. This value is 1 through 8 characters, and matches the name coded on the GRNAME parameter of the LUADD statements for the LUs. RACF requires this value to be in discrete form (that is, the value cannot contain any wildcard characters).

#### **partner-lu-name**

Is one of the following:

- The network name of the partner LU; that is, the 1- through 8-byte network-LU-name portion of their network-qualified names.
- The generic resource name associated with the partner LU. This value is 1 through 8 characters.

RACF accepts this value in generic form (that is, the value can contain wildcard characters).

After creating or changing APPCLU profiles, make sure that the APPCLU class has been activated using the SETROPTS CLASSACT(APPCLU) command. To cause VTAM to refresh its in-storage copies of the updated RACF APPCLU profiles for the LUs in a generic resource group, you must issue the MODIFY PROFILES command for each LU in the generic resource group. For example, if there are two LUs named APPCAP06 and APPCAP05 in generic resource group GENERNAM, issue MODIFY PROFILES,ID=APPCAP05 on the host where APPCAP05 resides, and MODIFY PROFILES,ID=APPCAP06 on the host where APPCAP06 resides.

If the partner LUs reside on different systems, you need a corresponding definition of the LUs on each partner's system. If RACF is installed on the partner system, define a corresponding APPCLU profile there, with the proper network ID and with the LU names in reverse order.

## Defining LU-to-LU Session Keys

For VTAM to verify LU-to-LU security, you need to specify an LU's session key in the APPCLU profile. The session key is a 1- through 16-digit hexadecimal value for the SESSKEY keyword, following the SESSION operand. For example:

```
RDEFINE APPCLU AA1.LU01.AA1.LU02 UACC(NONE) SESSION(SESSKEY(1234CD5))
```

If the partner LU is also on a RACF-protected system, you need to specify the same session key on the APPCLU profile for the partner LU; for example:

```
RDEFINE APPCLU AA1.LU02.AA1.LU01 UACC(NONE) SESSION(SESSKEY(1234CD5))
```

You can include other SESSION keywords to specify the following:

### **NOSESSKEY**

Delete an unneeded session key.

### **LOCK**

Lock a profile to prevent sessions from being established for this LU.

### **NOLOCK**

Unlock a locked profile to allow sessions to be established.

### **INTERVAL(n)**

Set an interval (the number of days the session key is valid) where *n* is in the range 1 through 32767 and does not exceed a global limit specified by the SETROPTS SESSIONINTERVAL command.

### **NOINTERVAL**

Specify no limit on the number of days the key is valid.

### **NOSESSION**

Delete the SESSION segment.

You can change existing APPCLU profiles using the RALTER command. For more information about specifying SESSION keywords on the RDEFINE or RALTER commands, see [z/OS Security Server RACF Command Language Reference](#).

When VTAM receives requests to establish a session with an LU that has an active session key, VTAM verifies that the requesting LU has a matching session key. If the requesting LU does not have a matching session key, VTAM and RACF send appropriate messages.

## Defining Conversation Security Levels that Sessions Allow

With VTAM 3.4 or higher, the CONVSEC field in the SESSION segment of the RACF APPCLU profile lets you specify the level of security that the local LU will accept from its partner LU. CONVSEC overrides the protection set by the SECACPT keyword of the VTAM APPL statement, which specifies the level of security allowed in conversation requests to an LU from anywhere in the network. CONVSEC narrows that level down to one allowed in a session between two specific LUs.

The CONVSEC values correspond to those of the SECACPT keyword:

### **Value**

**Means the local LU will accept:**

### **NONE**

Requests that contain no security information

### **CONV**

Requests with security information specified.

### ALREADYV

An indication that the user ID and password are already verified by the partner LU, and the partner is to be trusted (includes CONV).

### PERSISTV

Persistent verification (PV) requests. With PV, MVS verifies an inbound password the first time it arrives, then accepts the associated user ID without a password on subsequent Allocate requests in the same session (includes CONV).

### AVPV

Requests with user ID and password already verified and persistent verification indicators.

For example, to allow conversation requests that include security information or an already verified indicator, you could specify:

```
RDEFINE APPCLU AA1.LU02.AA1.LU01 SESSION(SESSKEY(1234CD5)
CONVSEC(ALREADYV))
```

To allow conversation requests that include security information or a persistent verification indicator, you could specify:

```
RDEFINE APPCLU AA1.LU02.AA1.LU01 SESSION(SESSKEY(1234CD5)
CONVSEC(PERSISTV))
```

To delete the conversation security parameters, you can specify NOCONVSEC on the RALTER command. NOCONVSEC tells RACF to ignore conversation security levels when sessions are being established between LUs, and defaults to the value of the SECACPT keyword on the APPL statement.

For more information about specifying conversation security parameters for APPCLU profiles, see [z/OS Security Server RACF Command Language Reference](#).

## Activating RACF Protection with APPCLU Profiles

When you are ready to start using the protection defined in the APPCLU profiles for each LU, the security administrator should activate the APPCLU class by issuing the RACF SETROPTS command on each system on which the APPCLU profile will be used. For example:

```
SETROPTS CLASSACT(APPCLU)
```

Any time an APPCLU profile is changed, use the following VTAM command to refresh the profile so that the change takes effect:

```
F procname,PROFILES,ID=local-lu-name
```

where *procname* represents the procedure name used to start VTAM. The profile changes do not take effect until you issue this command, and the sessions between the local and partner LUs have been terminated and are to be re-established. See [z/OS Communications Server: SNA Operation](#) for more information about acceptable values for *procname* on MODIFY commands.

When receiving inbound Allocate requests for conversations at an LU that has an active CONVSEC value, the system verifies that the conversation request contains allowed security parameters.

## Controlling the Use of VTAM ACBs

As described in Chapter 8, “Planning Sessions,” on page 95, one of the first steps in setting up an APPC/MVS environment is defining APPC LUs to VTAM by specifying them on VTAM APPL statements. Corresponding access method control blocks (ACBs) are then opened from the APPC address space when APPC is started on your MVS system.

Each LU has a name that is unique in the network. All requests for conversations with a particular TP include the name of the LU where the inbound TP resides. For example, in [Figure 87 on page 136](#), TPA specifies LU02 in its allocate request for a conversation with TPB.

To prevent non-APF-authorized programs from opening an ACB for a specific LU, or from registering as a member of a VTAM generic resource group, and thus perhaps intercepting requests addressed to that LU name, you can define the LU names in the RACF VTAMAPPL resource class with a universal access of NONE.

To create the RACF profiles and protect the APPC LUs, do the following:

1. Gather the names of the APPC LUs, as they are specified in the ACBNAME parameter on VTAM APPL statements and, if necessary, give them to your security administrator.
2. For each LU to be protected, the security administrator should create a RACF profile in the VTAMAPPL class, with the profile name matching the ACBNAME specified on the VTAM APPL statement, and give a universal access of NONE, for example:

```
RDEFINE VTAMAPPL acbname UACC(NONE)
```

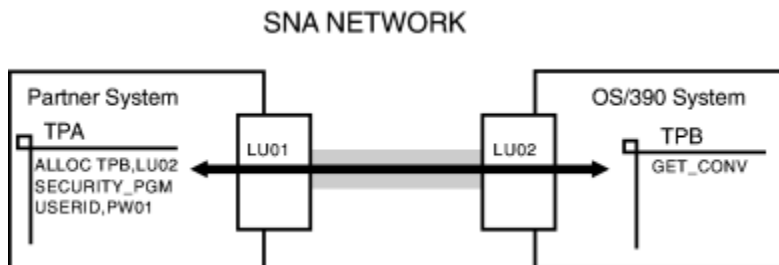
The ACB for that LU can then be opened only by APPC/MVS and other APF-authorized programs.

3. If the LUs are also members of a VTAM generic resource group, the security administrator should create a RACF profile in the VTAMAPPL class, with the profile name matching the generic resource name specified on the GRNAME parameter in LUADD statements for the LUs. For example:

```
RDEFINE VTAMAPPL generic-name UACC(NONE)
```

This VTAMAPPL definition protects against unauthorized use of only the generic resource name; it does not protect specific LUs in the generic resource group. To protect those LUs, you need to use the VTAMAPPL definition shown in Step “2” on page 144.

If you were creating VTAMAPPL profiles for the LUs shown in [Figure 89 on page 144](#), you would substitute LU02 for *acbname* in an RDEFINE command on that z/OS system. If the partner system was also protected by RACF, you could protect LU01 with a similar command on its system.



*Figure 89. Security for LU01 and LU02*

When you are ready to start using the protection defined in the VTAMAPPL profiles for each LU, the security administrator should activate the VTAMAPPL class and activate SETROPTS RACLIST processing for the class. For example:

```
SETROPTS CLASSACT(VTAMAPPL) RACLIST(VTAMAPPL)
```

Any time a VTAMAPPL profile is changed, SETROPTS RACLIST processing for the VTAMAPPL class must be refreshed for the change to take effect:

```
SETROPTS RACLIST(VTAMAPPL) REFRESH
```

## Conversation Security: Protecting APPC/MVS TPs

For APPC applications in which the outbound TP passes a user ID on an allocate request for an inbound TP on MVS, APPC/MVS provides the following conversation security mechanisms:

- Establishing a TP security environment on MVS
- Controlling user access to LUs
- Controlling user access from LUs

- Controlling user access to TP profiles and side information
- Controlling the ability to collect API trace data for conversations
- Obtaining TP work attributes from RACF user profiles
- Using persistent verification

The following sections describe these mechanisms in detail.

## Establishing a Security Environment for Inbound TPs on MVS

When an inbound allocate request for a TP on MVS has a security type of SAME or PGM and includes a user ID that is defined to RACF, APPC/MVS automatically uses the RACF user profile for that ID to create the security environment for the TP to run in. The TP can then access any data or resources that the user is allowed to access. The RACF profile can also provide individualized SYSOUT and accounting information for the TP to use. If there is no RACF profile available for the user ID, the inbound allocate request is rejected.

Figure 90 on page 145 shows how APPC/MVS uses the RACF profile to establish a security environment when allocate requests include a user ID for which there is a RACF user profile on MVS.

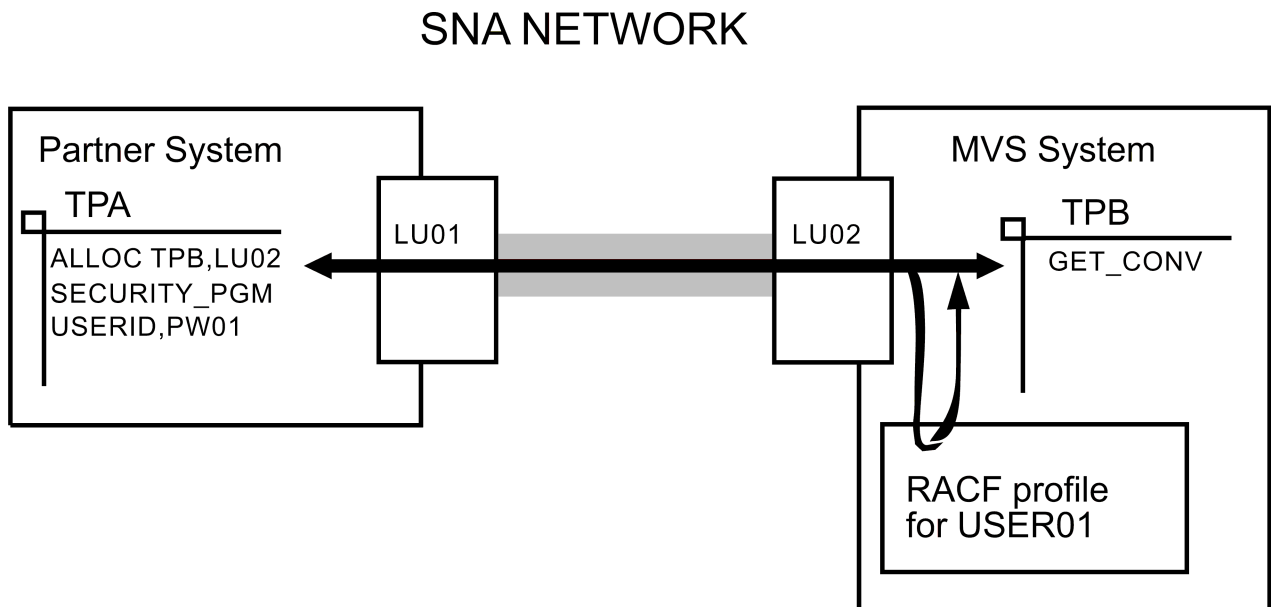


Figure 90. Setting Security Environment from the RACF Profile

For all inbound TPs on MVS, the security administrator must ensure that a RACF user profile exists for each user ID that the outbound TP might pass on the allocate request. If the outbound TP also passes a security profile name, the security administrator must also:

1. Create a RACF group with that name
2. Connect the user to the group.

When an inbound TP on MVS is allocated with a security type of NONE, the inbound TP runs without a user ID and can only access resources that are available with universal access.

## Controlling User Access to LUs

One of the conversation security mechanisms you can implement is controlling access, by user ID, to specific APPC/MVS LUs. Such control is useful when an LU represents a group of related TPs or a transaction scheduler. Through the APPL class, you can control access to an LU through one of two ways:

1. By granting access to only specific users or groups

This method provides the most restrictive security for the LU, because you begin by prohibiting **any** access to the LU, and then gradually grant access to specific users on an as-needed basis. Depending

on how your installation has defined security profiles for users or groups, and when you determine a user's need to access the LU, this method might require frequent updates to the LU's security information.

## 2. By prohibiting security\_none Allocate requests

This method provides security for the LU by accepting only those Allocate requests associated with a user ID. As in the first method, you begin by prohibiting any access to the LU, but then grant access to all user IDs at once, through only two commands. In effect, this method prohibits requests with a security\_type of security\_none from entering the system.

You may use either method to control access to individual LUs, or to all LUs in a VTAM generic resource group. If an LU is a member of a generic resource group, you must use its generic resource name, instead of its specific name, on the RDEFINE command for the APPL class.

Also, you may use RACF variables in the APPL definitions, to simplify the task of controlling user access to LUs.

## Granting Access to Only Specific Users or Groups

Granting access to an LU by specific user or group IDs consists of the following steps:

### 1. Defining the LU profile in a RACF APPL class (RDEFINE command)

Use profiles in the RACF APPL class to define which local user IDs may allocate conversations with TPs on an MVS LU. APPL profiles use the same 1- through 8-character name as the local LU specified on the VTAM APPL statement. For example:

```
RDEFINE APPL luname UACC(NONE)
```

To protect the MVS LU shown in [Figure 90 on page 145](#), for example, you could specify:

```
RDEFINE APPL LU02 UACC(NONE)
```

Specifying RDEFINE with UACC(NONE) prohibits anyone from accessing that LU.

### 2. Allowing access to the LU by user or group ID (PERMIT command)

To allow access to the LU, enter the PERMIT command to grant READ access to the LU for specific users or groups. For example,

```
PERMIT LU02 CLASS(APPL) ID(userid or groupid) ACCESS(READ)
```

### 3. Activating the changes to an APPL class profile (SETROPTS command)

When you are ready to start using the protection defined in the APPL profiles for each LU, the security administrator should activate the APPL class and activate SETROPTS RACLIST processing for the class. For example:

```
SETROPTS CLASSACT(APPL) RACLIST(APPL)
```

Any time an APPL profile is changed, SETROPTS RACLIST processing for the APPL class must be refreshed for the change to take effect:

```
SETROPTS RACLIST(APPL) REFRESH
```

After you issue RDEFINE, PERMIT, and SETROPTS as illustrated, APPC/MVS verifies that all inbound allocate requests addressed to LU02 are permitted to access it. APPC/MVS verifies the user ID, password, and security profile, if any, provided on the request.

## Prohibiting Security\_None Allocate Requests

Protecting an LU by prohibiting security\_none Allocate requests consists of these steps:

### 1. Defining the LU profile in a RACF APPL class (RDEFINE command)



As in Step “1” on page 146 under “Granting Access to Only Specific Users or Groups” on page 146, use this RDEFINE command for LU02:

```
RDEFINE APPL LU02 UACC(NONE)
```

Specifying RDEFINE with UACC(NONE) prohibits anyone from accessing that LU.

## 2. Allowing access to the LU (PERMIT command)

Again, as in Step “2” on page 146 under “Granting Access to Only Specific Users or Groups” on page 146, use a PERMIT command for LU02 with one key difference: the value specified for ID.

```
PERMIT LU02 CLASS(APPL) ID(*) ACCESS(READ)
```

Specifying ID(\*) allows only Allocate requests with a security\_type of security\_pgm or security\_same to be accepted for this LU.

## 3. Activating the changes to an APPL class profile (SETROPTS command)

As in Step “3” on page 146 under “Granting Access to Only Specific Users or Groups” on page 146, the security administrator should activate the APPL class and activate SETROPTS RACLIST processing for the class by issuing:

```
SETROPTS CLASSACT(APPL) RACLIST(APPL)
```

Any time an APPL profile is changed, SETROPTS RACLIST processing for the APPL class must be refreshed for the change to take effect:

```
SETROPTS RACLIST(APPL) REFRESH
```

After you issue RDEFINE, PERMIT, and SETROPTS as illustrated, APPC/MVS rejects all inbound requests for that LU that have a security type of security\_none because those requests are not associated with a user ID.

## Controlling User Access to LUs in a VTAM Generic Resource Group

The procedures listed in “Granting Access to Only Specific Users or Groups” on page 146 and “Prohibiting Security\_None Allocate Requests” on page 146 illustrate how to control user access to an individual LU. To control access to LUs that are members of the same VTAM generic resource group, the procedures are the same, except for the name you specify. Instead of using the LU name that matches the local LU name on the VTAM APPL statement, specify the same name as the generic resource name on the GRNAME parameter on the LUADD statements for the LUs. For example:

```
RDEFINE APPL generic-name UACC(NONE)
PERMIT generic-name CLASS(APPL) ID(userid or groupid) ACCESS(READ)
```

## Using RACF Variables for the APPL Class

Instead of issuing a set of RACF commands for each individual LU, as illustrated in “Granting Access to Only Specific Users or Groups” on page 146 and “Prohibiting Security\_None Allocate Requests” on page 146, you may use RACF variables to secure multiple LUs with a single set of commands. For example, the following commands use a RACF variable for multiple LUs that are not part of a VTAM generic resource group:

```
RDEFINE RACFVARS &LUS UACC(NONE) ADDMEM(LLU1 LLU2 LLU3)
RDEFINE APPL &LUS UACC(NONE)
PERMIT &LUS CLASS(APPL) ID(USER1) ACCESS(READ)
PERMIT &LUS CLASS(APPL) ID(USER2) ACCESS(READ)
:
SETROPTS CLASSACT(APPL RACFVARS) RACLIST(RACFVARS)
```

You may use RACF variables for LUs in a VTAM generic resource group as well; simply use the generic resource name as a value for the ADDMEM operand.

Any time a RACFVARS profile is changed, SETROPTS RACLIST processing for the RACFVARS class must be refreshed for the change to take effect.

## Controlling User Access from LUs

You can further control a user's access to APPC/MVS LUs by controlling which LU the user's request can come from.

Use RACF profiles in the APPCPORT class to define which user IDs may access the system from a given LU (APPC port of entry). APPCPORT profile names are of the form partner-lu-name, where partner-lu-name is the locally known name of the partner LU (1 through 8 characters). For example:

```
RDEFINE APPCPORT luname UACC(NONE)
PERMIT luname CLASS(APPCPORT) ID(userid or groupid) ACCESS(READ)
```

If the APPCPORT class is active, APPC/MVS requires that the user have at least READ access to the APPCPORT profile in order to access the system.

Look again at [Figure 90 on page 145](#). To permit USER01 to initiate MVS TPs such as TPB by request from LU01, you could use the following definition on LU02's system:

```
RDEFINE APPCPORT LU01 UACC(NONE)
PERMIT LU01 CLASS(APPCPORT) ID(USER01) ACCESS(READ)
```

When you are ready to start using the protection defined in the APPCPORT profiles for each LU, the security administrator should activate the APPCPORT class and activate SETROPTS RACLIST processing for the class. For example:

```
SETROPTS CLASSACT(APPCPORT) RACLIST(APPCPORT)
```

Any time an APPCPORT profile is changed, SETROPTS RACLIST processing for the APPCPORT class must be refreshed for the change to take effect:

```
SETROPTS RACLIST(APPCPORT) REFRESH
```

## Controlling User Access to TP Profiles and Side Information on MVS

On MVS, side information and TP profiles contain routing and scheduling information that MVS uses to find and initiate TPs in response to allocate requests from other TPs. These TP profiles are distinct from RACF profiles.

APPC/MVS administrators on MVS must create the TP profiles and side information before users can invoke the TPs named in the TP profiles and side information. Special security mechanisms let you control access to side information and TP profiles on MVS. By controlling access to TP profiles on MVS, you control access to the TPs themselves. [Figure 91 on page 149](#) shows the role of side information and TP profiles in establishing conversations.

To illustrate the use of side information on MVS, [Figure 91 on page 149](#) shows TPA on an z/OS system. In this case, TPA uses a symbolic destination name (SYMDES1) to identify the inbound TP, and APPC/MVS checks the side information file to determine the actual names of the inbound TP and LU. [Figure 91 on page 149](#) also shows a TP profile for TPB. The TP profile contains scheduling information that MVS uses to initiate TPB.

## SNA NETWORK

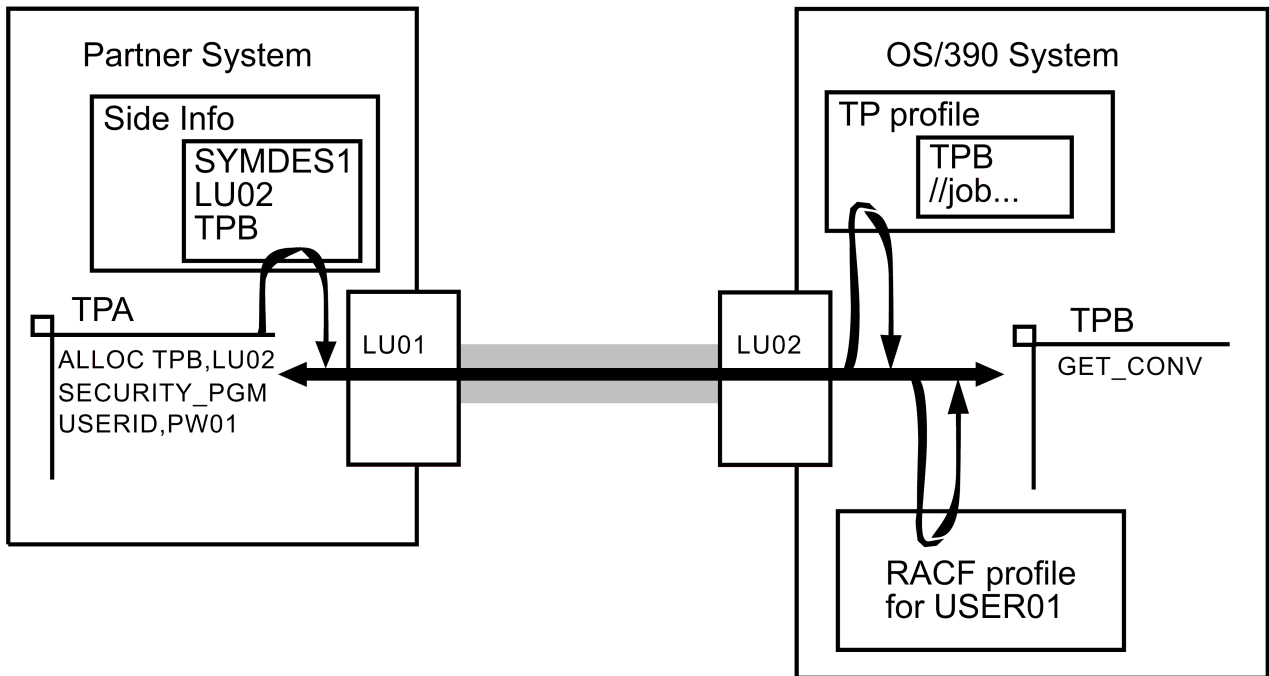


Figure 91. TP Profiles and Side Information

APPC administrators create and maintain TP profiles and side information by using the APPC/MVS administration utility (ATBSDLFUMU), or the APPC/MVS administration dialog (an interactive front-end to ATBSDLFUMU). The TP profiles and side information entries are stored in VSAM key-sequenced data sets (KSDS). To protect a KSDS and its individual entries, do the following:

- Define a data set profile for the KSDS with UACC(NONE), then give ATBSDLFUMU program access to the RACF profile
- Use the APPC/MVS administration utility or dialog to create database security tokens (database tokens) to associate with the data set
- Create RACF profiles in the APPCTP and APPCSI classes to control access to individual entries (TP profiles and side information entries) in each KSDS.

### Giving Program Access to the APPC/MVS Administration Utility

To ensure that TP profiles and side information files are accessed only through the APPC/MVS administration utility (ATBSDLFUMU), the system security administrator may use the program access to data sets (PADS) function of RACF for the data sets specified in the SYSSDLIB DD statement. For a PADS environment, the administrator must define certain programs to the RACF PROGRAM class; those programs vary, depending on the method used to invoke the utility:

- For a batch job, define the following:
  - ATBSDLFUMU entry points ATBINMIG, ATBSDEPE, ATBSDLFUMU, ATBSDLFCS, and ATBSDLFM1
  - SYS1.LINKLIB members that ATBSDLFUMU calls to check the syntax of JCL
- For an application program, define the programs listed for a batch job, and any programs that are loaded before the ATBSDLFUMU utility is invoked.
- For a REXX program, define the programs listed for a batch job, the TSO/E Information Center Facility program ICQASLIO, and any programs that are loaded before the ATBSDLFUMU utility is invoked.

For example, to give administrators in the ADMIN01 group access, use the following commands:

```
RDEFINE PROGRAM ATBINMIG ADDMEM('SYS1.MIGLIB'/volser) UACC(NONE)
RDEFINE PROGRAM ATBSDEPE ADDMEM('SYS1.MIGLIB'/volser) UACC(NONE)
```

```
RDEFINE PROGRAM ATBSDFMU ADDMEM('SYS1.MIGLIB'/volser) UACC(NONE)
RDEFINE PROGRAM ATBSDFCS ADDMEM('SYS1.MIGLIB'/volser) UACC(NONE)
RDEFINE PROGRAM ATBSDFM1 ADDMEM('SYS1.MIGLIB'/volser) UACC(NONE)
RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'/volser/NOPADCHK) UACC(NONE)
PERMIT ATBSDFMU CLASS(PROGRAM) ID(ADMIN01) ACCESS(READ)
ADDSD 'data.set.name' GENERIC UACC(NONE)
PERMIT 'data.set.name' ID(ADMIN01) WHEN(PROGRAM(ATBSDFMU)) ACCESS(UPDATE)
```

If you encounter messages ATB369I or ICH408I after defining these programs, follow the procedure in [z/OS Security Server RACF Diagnosis Guide](#) for obtaining traces for PADS errors. This procedure helps identify additional programs that require definition to the RACF PROGRAM class.

If the APPC/MVS administration dialog is used as the interface to the utility, and PADCHK is specified in any of the members defined in the PROGRAM class profile, then all programs that are loaded under the TCB must be included in the conditional access list for all data sets being opened. Additionally, ICQASLIO must be in the conditional access list of any data sets being opened. The following command may be used:

```
PERMIT dataset_profile_name ID (ADMIN01) WHEN(PROGRAM(ICQASLIO))
ACCESS(UPDATE)
```

**Note:** Program control must be active on the system for this access control to take effect. For more information about controlling program access to data sets, see [z/OS Security Server RACF Security Administrator's Guide](#).

## Controlling Access to Database Tokens

Each data set containing TP profiles or side information entries can have a database token assigned to it. The database token is a 1 through 8-character name from character set Type A that represents the file name in security definitions. If a TP profile or side information entry has no database token, APPC/MVS does not call RACF to verify access requests.

To create and maintain database tokens, APPC/MVS administrators use the APPC/MVS administration utility's DBMODIFY command. For more information about that command, see [Chapter 6, “Using the APPC/MVS Administration Utility,”](#) on page 73.

To control administrator access to database tokens, you can define the APPCMVS.DBTOKEN profile in RACF's FACILITY class and give UPDATE access to the appropriate APPC/MVS administrators. For example:

```
RDEFINE FACILITY APPCMVS.DBTOKEN UACC(NONE)
PERMIT APPCMVS.DBTOKEN CLASS(FACILITY) ID(userid or groupid) ACCESS(UPDATE)
```

## Controlling User Access to Side Information

Authority to administer individual entries in the side information file is provided by RACF profiles in the APPCSI class. APPCSI profile names are of the form dbtoken.SYS1.symbolic-destination-name, where:

- dbtoken is the database token associated with the side information file (1 through 8 characters).
- symbolic\_destination\_name is the symbolic destination name (1 through 8 characters) associated with the side information entry.

For example:

```
RDEFINE APPCSI dbtoken.SYS1.symdname UACC(NONE)
```

APPC/MVS administrators need READ access to view side information entries and UPDATE access to create, modify, and delete side information entries.

For example, assuming TPA in Figure 91 on page 149 is on MVS, and the side information file has a database token of TOKEN1, you could use the following commands to permit ADMIN01 to view the entry for SYMDES1:

```
RDEFINE APPCSI TOKEN1.SYS1.SYMDES1 UACC(NONE)
PERMIT TOKEN1.SYS1.SYMDES1 CLASS(APPCSI) ID(ADMIN01) ACCESS(READ)
```

You could use the following command to allow ADMIN01 to modify the entry for SYMDES1, for example, when moving TPB to another LU:

```
PERMIT TOKEN1.SYS1.SYMDES1 CLASS(APPCSI) ID(ADMIN01) ACCESS(UPDATE) .
```

Users who use symbolic destination names on outbound allocate requests do not require access to APPCSI profiles.

When you are ready to start using the protection defined in the APPCSI profiles, the security administrator should activate the APPCSI class and activate SETROPTS RACLIST processing for the class. For example:

```
SETROPTS CLASSACT(APPCSI) RACLIST(APPCSI)
```

Any time an APPCSI profile is changed, SETROPTS RACLIST processing for the APPCSI class must be refreshed for the change to take effect:

```
SETROPTS RACLIST(APPCSI) REFRESH
```

## Controlling User Access to TPs

For each MVS image on which APPC/MVS is started, your installation can define one or more **TP profiles** for a given TP. These profiles may have different levels (SYSTEM, GROUPID, or USERID), as defined in “Creating a TP Profile” on page 59. By defining **RACF profiles** in the RACF APPCTP class, and then permitting access to those RACF profiles, you can control which user IDs may access the APPC/MVS TP profiles and execute the associated TPs.

### Defining RACF APPCTP Profiles

APPCTP profile names are of the form dbtoken.level.tpname, where:

- dbtoken is the database token associated with the TP profile file (1 through 8 characters).
- level is one of the following:
  - SYS1 if the transaction program is available to all users who can access the LU.
  - The group ID if the transaction program is available to a group.
  - The user ID if the transaction program is for a specific user.
- tpname is the name of the transaction program (1 through 64 characters from the 00640 character set, or the Type A character set, or 7-9 characters for an SNA service TP.)

For example:

```
RDEFINE APPCTP dbtoken.level.tpname UACC(NONE)
```

If tpname contains periods, RACF treats them as qualifiers. For a TP called JOE.MAIL.PGM, you could have a profile such as:

```
RDEFINE APPCTP dbtoken.level.JOE.MAIL.PGM UACC(NONE)
```

If generic security checking is active for the APPCTP class, asterisks act as generic characters anywhere in the APPCTP resource name. For example, to create one APPCTP profile for all TP names beginning with JOE for a given database token and level, you could specify:

```
RDEFINE APPCTP dbtoken.level.JOE* UACC(NONE)
```

To create one APPCTP profile for all system-level TP profiles that have a database token of TOKEN1, you could specify:

```
RDEFINE APPCTP TOKEN1.SYS1* UACC(NONE)
```

### ***Permitting Access to RACF APPCTP Profiles***

APPC/MVS administrators need READ access to view TP profiles and UPDATE access to create, modify, and delete TP profiles. APPC users need EXECUTE access to a TP profile to run the associated transaction program. For example, assume TPB in [Figure 91 on page 149](#) is in a system-level TP profile with a database token of TOKEN2, protected with the following command:

```
RDEFINE APPCTP TOKEN2.SYS1.TPB UACC(NONE)
```

- To give administrator ADMIN01 access to view the contents of the TP profile for TPB:

```
PERMIT TOKEN2.SYS1.TPB CLASS(APPCTP) ID(ADMIN01) ACCESS(READ)
```

- To give administrator ADMIN02 access to change the contents of the TP profile for TPB, for example, to add some JCL:

```
PERMIT TOKEN2.SYS1.TPB CLASS(APPCTP) ID(ADMIN01) ACCESS(UPDATE)
```

- To give user USER01 access to run TPB:

```
PERMIT TOKEN2.SYS1.TPB CLASS(APPCTP) ID(USER01) ACCESS(EXECUTE)
```

To protect TP profiles and the inbound TPs they represent, collect the following information and, if necessary, give it to your security administrator:

- A list of all TP profiles to be protected, in the form dbtoken.level.tpname
- A list of user IDs of APPC users needing EXECUTE access to each TP profile
- A list of user IDs of APPC/MVS administrators needing READ or UPDATE access to each TP profile.

The security administrator should create an APPCTP profile for each of the TP profiles, using generic characters where appropriate, and give the appropriate access to each user.

When you are ready to start using the protection defined in the APPCTP profiles, the security administrator should activate the APPCTP class and activate SETROPTS RACLIST processing for the class. For example:

```
SETROPTS CLASSACT(APPCTP) RACLIST(APPCTP)
```

Any time an APPCTP profile is changed, SETROPTS RACLIST processing for the APPCTP class must be refreshed for the change to take effect.

```
SETROPTS RACLIST(APPCTP) REFRESH
```

To protect TPs that do not have a TP profile, define an APPCTP profile that has a level of SYS1.

### ***Understanding Access Checking of an Inbound Allocate Request for a TP***

The values you specify in RDEFINE and PERMIT commands for APPCTP profiles affect how the system verifies user access, but in combination with the values in the APPC/MVS TP profiles, the RACF APPCTP profile, and the APPCPMxx parmlib member. To determine which values are most efficient, you need a general understanding of how the system performs security checks. When an inbound Allocate request for a TP arrives, the system:

1. Verifies the user, through the Userid and Password parameter values on the Allocate request.

2. Searches for the most restrictive RACF profile for which the verified user has EXECUTE or higher authority. To accomplish this, the system searches TP profiles in the following order, beginning with the level specified in the APPCPMxx parmlib member:
  - a. USER level TP profiles
  - b. GROUP level TP profiles
  - c. SYSTEM level TP profiles

For USER level TP profiles, the system compares the Userid parameter value on the Allocate request with the RACF user ID verified in Step “1” on page 152.

For GROUP level TP profiles, the system compares the Profile parameter value on the Allocate request with the group ID value specified in the TP profile. If both the local and partner TPs run on MVS, these values can, but do not have to, represent a RACF group. If the Allocate request does not contain a Profile parameter value, the GROUP level TP profiles cannot be used.

## Protecting Multi-Trans TP Profiles

There are special security considerations for TPs that have a TP schedule type of multi-trans. Multi-trans TPs are inbound TPs on MVS that run on behalf of multiple users in sequence and remain active between conversations. In each conversation, multi-trans TPs run in a security environment based on the caller's user ID or group ID, if passed.

Moreover, during initialization, multi-trans TPs run under a user ID that is specified in the GENERIC\_ID keyword of the TP profile. Therefore, the generic user ID also must be permitted to access any RACF-protected resources that the multi-trans TP might need while running under that ID. For example, if the TP is running under the generic user ID and an inbound allocate request arrives from an LU protected by an APPCPORT profile, the generic user ID must be permitted to that resource. Likewise, if a multi-trans TP is running under a generic user ID and allocates a conversation to an LU protected by an APPL profile, the generic user ID must be permitted to that resource.

### Protecting the Generic User ID at Installation Time

Because the generic user ID applies to processing that must be isolated from the different conversation users, the generic user ID must be secure from unauthorized specification and modification in the TP profile. At the time of installation, your RACF administrator must use the APPCMVS.TP.MULTI.genusrid profile in the RACF FACILITY class to protect the generic user ID and control access to TP profiles for multi-trans TPs. The command shown in Figure 92 on page 153 prevents all users from adding or modifying multi-trans TP profiles with any generic user ID.

```
RDEFINE FACILITY APPCMVS.TP.MULTI.* UACC(NONE)
```

*Figure 92. Protecting the Generic User ID*

After issuing the RDEFINE command shown in Figure 92 on page 153, your RACF administrator can permit individual users to define multi-trans profiles with a specific user ID. For example, if Bob needs to create a multi-trans TP profile with the generic user ID ADMIN, he must first get permission from a RACF administrator. Your RACF administrator would then issue the command shown in Figure 93 on page 153.

```
RDEFINE FACILITY APPCMVS.TP.MULTI.ADMIN UACC(NONE)  
PERMIT APPCMVS.TP.MULTI.ADMIN CLASS(FACILITY) ID(BOB) ACCESS(UPDATE)
```

*Figure 93. Permitting Update Access to a Multi-trans TP*

After receiving RACF UPDATE access, Bob can use the TPADD command to create a multi-trans profile with a generic user ID of ADMIN. For more information about the TPADD command see Chapter 6, “Using the APPC/MVS Administration Utility,” on page 73.



## ***Granting the Appropriate Level of Access to a Multi-Trans TP***

Access to multi-trans TP profiles should be limited to administrators who have the same authority as the security administrator or as a system programmer who is responsible for updating authorized libraries. These administrators need READ access to view TP profiles for multi-trans TPs and UPDATE access to create, update, or delete TP profiles for multi-trans TPs. Note that to run a multi-trans TP you only need EXECUTE access to the RACF profile for the TP in the APPCTP class. For more information about providing execute access, see [“Controlling User Access to TPs” on page 151](#).

## **Controlling Ability to Collect API Trace Data**

The APPC/MVS application programming interface (API) trace facility allows programmers to more easily diagnose problems with APPC/MVS TPs, by collecting data about APPC/MVS or CPI-C calls that an APPC/MVS TP issues. Through the ATBTRACE REXX exec, programmers can start tracing activity for only specific TPs or users, or for many TPs, many conversations, and many users. They can stop or list the status of tracing activity through the exec, as well.

Security-related issues for the API trace facility include:

- Restricting access to API trace resources
- Allowing a system administrator to control API tracing activity
- Restricting API tracing activity to specific users or conversations
- Allowing API tracing activity for security\_none conversations.

If necessary, see [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#) for more information about using the API trace facility.

## **Restricting Access to API Trace Resources**

Your installation might want to restrict access to API trace resources to prevent unauthorized users from viewing service calls and data passed between critical transaction programs. Also, depending on the number of active API traces, the performance of APPC/MVS TPs might be adversely affected. In a test system, this possible performance impact might be acceptable; however, in a production system, performance degradation is not acceptable. To limit the possible performance impact, and to control access to sensitive data transmitted between TPs, consider using the RACF FACILITY class to restrict access to API trace data.

Using RACF FACILITY class profiles, your installation may control which conversations may be traced, and which users are allowed to start and stop API traces. The class profile values for LU and TP should match the LU and TP values specified on ATBTRACE requests to start or stop tracing. To do so:

1. Define a FACILITY class profile for the resource ATBTRACE.netid.lu\_name.tp\_name in which:

### **ATBTRACE**

Identifies the high-level qualifier for the API trace security resource.

### **netid**

Identifies the network ID portion of the network-qualified LU name. To collect trace data for local TPs, use the ID of the local network. For tracing to begin, ATBTRACE START requests for these local TPs may contain either an unqualified or network-qualified name of a local LU. To collect trace data for remote TPs, use the ID of the remote network. For tracing to begin, ATBTRACE START requests for the remote TPs must specify the network-qualified name of the remote LU. To successfully collect trace data for both local and remote TPs, you must define more than one FACILITY class profile: one with the local network ID, and another with the remote network ID.

### **lu\_name**

Identifies the network-LU-name portion of the network-qualified LU name, where the TP identified by *tp\_name* will run.

### **tp\_name**

Identifies the TP whose conversation is to be traced.

2. Grant READ access to users that need the ability to start and stop API traces.



3. If necessary, refresh SETROPTS RACLIST processing for the FACILITY class, for the change to take effect.

If the FACILITY class is active, APPC/MVS requires a security profile that protects the ATBTRACE.netid.lu\_name.tp\_name resource; APPC/MVS rejects a START or STOP request if either of the following conditions are true:

- The FACILITY class profile for ATBTRACE is not defined.
- The user submitting the START or STOP request does not have access to the profile.

For general information about the RACF FACILITY class, see [z/OS Security Server RACF Security Administrator's Guide](#). For RACF command options and operands, see [z/OS Security Server RACF Command Language Reference](#).

## Allowing a System Administrator to Control API Tracing Activity

IBM recommends that the installation allow a system administrator to control API tracing activity on each z/OS system, even when tracing is restricted to only a few other users or for only a limited number of conversations. For example, to allow only tracing START and STOP requests issued from user ID ADMIN1, for all conversations for all TPs that run on any LU in the system, use these commands:

```
RDEFINE FACILITY ATBTRACE.*.*.* UACC(NONE)
PERMIT ATBTRACE.*.*.* CLASS(FACILITY) ID(ADMIN1) ACCESS(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

These commands prevent any trace activity on the system; you could use them for a production system, changing the access only when an error occurs and you want to collect trace data to diagnose the problem.

## Restricting API Tracing Activity to Specific Users or Conversations

The following sets of RACF commands illustrate several variations of restricting the ability to collect API trace data.

- To allow any user to start or stop tracing for any conversation on the system:

```
RDEFINE FACILITY ATBTRACE.*.*.* UACC(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

- To allow any user to start or stop tracing for only those conversations for the TP named COMPANY.MAIL that runs on the LU named NET01.LU01:

```
RDEFINE FACILITY ATBTRACE.NET01.LU01.COMPANY.MAIL UACC(NONE)
PERMIT ATBTRACE.NET01.LU01.COMPANY.MAIL CLASS(FACILITY) ID(*)
ACCESS(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

- To allow only tracing START and STOP requests issued from user ID JOE, for only those conversations for the TP JOE.MAIL.PGM running on LU NET02.LU02:

```
RDEFINE FACILITY ATBTRACE.NET02.LU02.JOE.MAIL.PGM UACC(NONE)
PERMIT ATBTRACE.NET02.LU02.JOE.MAIL.PGM CLASS(FACILITY) ID(JOE)
ACCESS(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

In this case, no other user is allowed to start or stop trace for these conversations.

- To allow only tracing START and STOP requests issued from user ID FRED, for only those conversations for the TP FRED.MAIL.PGM running in network NET02, on any LU in the VTAM generic resource group GEN02:

```
RDEFINE FACILITY ATBTRACE.NET02.GEN02.FRED.MAIL.PGM UACC(NONE)
PERMIT ATBTRACE.NET02.GEN02.FRED.MAIL.PGM CLASS(FACILITY) ID(FRED)
```

```
ACCESS(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

- To allow only tracing START and STOP requests issued from user ID MYUSER, for all conversations for all TPs that run on LU NET99.LU99:

```
RDEFINE FACILITY ATBTRACE.NET99.LU99.* UACC(NONE)
PERMIT ATBTRACE.NET99.LU99.* CLASS(FACILITY) ID(MYUSER) ACCESS(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

## Allowing API Tracing Activity for Security\_None Conversations

When APPC/MVS receives inbound Allocate requests with a Security\_type of security\_none, you have to grant access to the FACILITY class only when ATBTRACE START and STOP requests are invoked from either:

- The TP's profile JCL, or
- A call issued by the TP itself.

For example, suppose an APPC/MVS TP named COMPANY.NEWS, which runs on LU NET01.LU01, invokes the ATBTRACE REXX exec to start tracing. For this START request to be successful, issue the following commands before APPC/MVS schedules the TP in response to an Allocate with a Security\_type of security\_none:

```
RDEFINE FACILITY ATBTRACE.NET01.LU01.COMPANY.NEWS UACC(READ)
SETROPTS CLASSACT(FACILITY) RACLIST(FACILITY)
SETROPTS RACLIST(FACILITY) REFRESH
```

## Obtaining SYSOUT and Account Information from RACF User Profiles

RACF allows you to specify SYSOUT and account number information in a RACF user profile, to be used when a TP is run on behalf of that user. To store the SYSOUT and account information in the RACF user profile, specify any of the following keywords on the WORKATTR segment of the RACF ADDUSER and ALTUSER commands. Include single quotes around any values containing blanks.

### WANAME

Specifies the name of the user that SYSOUT information is to be delivered to. Up to sixty EBCDIC characters may be specified.

### WABLDG

Specifies the building that SYSOUT information is to be delivered to. Up to sixty EBCDIC characters may be specified.

### WADEPT

Specifies the department that SYSOUT information is to be delivered to. Up to sixty EBCDIC characters may be specified.

### WAROOM

Specifies the room that SYSOUT information is to be delivered to. Up to sixty EBCDIC characters may be specified.

### WAADDR1 through WAADDR4

Specifies up to four more address lines for SYSOUT delivery. Up to sixty EBCDIC characters may be specified for each line.

### WAACCNT

Specifies an account number for APPC/MVS processing. Up to 255 EBCDIC characters may be specified.

**Note:** The WAACCNT value overrides an account number in the job statement.

For example, you could use a command such as the following to add SYSOUT and account information to the RACF profile for user ID USER01:

```
ALTUSER USER01 WORKATTR (WANAME('SAMMY CHARLES') WABLDG('BUILDING 999')  
WADEPT('DEPARTMENT 101A') WAROOM('ROOM 1001') WAADDR1('POKTOWN NY')  
WAACCN1(549300PFG10))
```

For more information about specifying SYSOUT and account information in a RACF user profile, including syntax of the RACF ADDUSER command, see [z/OS Security Server RACF Command Language Reference](#).

## Extracting SYSOUT and Account Information from RACF User Profiles

For any inbound TPs on MVS that require personalized SYSOUT and account information for each user, you need to tell APPC to extract that information from the user's RACF profile. To cause APPC/MVS to extract the SYSOUT and account information from the user's RACF profile, an APPC/MVS administrator must use the APPC/MVS administration dialog or utility to specify YES on the TAILOR\_SYSOUT and TAILOR\_ACCOUNT keywords for the appropriate TP. For more information about those keywords, see Chapter 6, “Using the APPC/MVS Administration Utility,” on page 73.

If you don't use these procedures to extract SYSOUT and account information from the RACF profile, any SYSOUT or account information specified in the TP profile will be in effect for all users of the TP.

Because APPC/MVS administrators can best provide the necessary information, the security administrator can give APPC/MVS administrators update access to WORKATTR segments in RACF user profiles through RACF's field-level access checking.

---

### Programming Interface Information

---

## Using Persistent Verification (PV)

With VTAM 3.4 or later and RACF installed, APPC/MVS can receive persistent verification requests. Persistent verification (PV) is an option that LUs can specify on outbound allocate (attach) requests. Persistent verification is an application security mechanism that maintains lists of verified user IDs and reduces the flow of passwords across the network.

APPC/MVS LUs can receive but cannot send persistent verification requests.

To use persistent verification on your system, you must:

- Allow it between the appropriate LUs by specifying the PERSISTV or AVPV values in either:
  - The VTAM APPL statement's SECACPT keyword, as described in [“Specifying the Level of Conversation Security for VTAM”](#) on page 138, **or**
  - The RACF APPCLU profile SESSION segment, through the CONVSEC keyword, as described in [“Defining Conversation Security Levels that Sessions Allow”](#) on page 142.

If you specify conversation security values in **both** the VTAM APPL statement and the RACF APPCLU profile, the RACF value overrides the VTAM value.

- Make sure the RACF subsystem is available to handle persistent verification requests. Refer to [z/OS Security Server RACF System Programmer's Guide](#) for more information about the RACF subsystem.
- Make sure the network LU names of participating LUs are unique across interconnected networks. If your installation is using APPC/MVS support of network-qualified names to ensure that LU names are unique, the results of persistent verification requests are unpredictable.

A network-qualified name is a 17-byte name in the form *network\_id.network\_lu\_name*, where the network-ID portion uniquely identifies an LU when the network-LU-name portion is identical to LU names on other networks in the installation. With persistent verification, the network ID is not used to verify PV requests. If network LU names themselves are not unique, LUs might, for example, accept conversations from the wrong partner LUs.

Because of this unpredictability, IBM does not recommend the use of APPC/MVS support for network-qualified names for those LUs that your installation uses for persistent verification requests.

On sessions that allow persistent verification, the following sequence of events establishes persistent verification for user IDs passed on that session:

- The outbound LU adds a user ID to its own signed-on-to list and sends security information like a user ID and password on an outbound allocate request, with a PV **sign-on** indicator in the function management header 5 (FMH-5).
- If the inbound system (possibly MVS) successfully verifies the security information in the FMH-5 sent from the partner LU, the inbound system enters the user ID in a "signed-on-from list". The signed-on-from list contains a list of user IDs signed on for persistent verification from that outbound LU.
- On subsequent allocate requests containing that user ID and a PV **signed-on** indicator, information that validates the user's identity is not included.
- The inbound system receives the subsequent allocate requests and verifies that the user is in the signed-on-from list.
- The user ID normally remains on the list for the duration of the session. Operator action or communication failure can cause the user ID to be dropped from the list.

MVS operators can use the RACF #DISPLAY SIGNON command to display the signed-on-from list and the #SIGNOFF command to remove entries from the signed-on-from list while a session is active.

For example, the following command would search the signed-on-from list for user USER01 to see if that user is signed on for persistent verification from LU01. If USER01 is signed on from that specific port of entry (POE), a list is returned to the operator console.

```
#DISPLAY SIGNON,APPL(LU02),POE(LU01),USER(USER01),GROUP(*)
```

The following operator command would remove user USER01 from the signed-on-from list of LU02:

```
#SIGNOFF APPL(LU02),POE(LU01),USER(USER01),GROUP(*)
```

See [z/OS Security Server RACF Command Language Reference](#) for more information about these commands.

When the operator removes an entry from the signed-on-from list, RACF grants control to the persistent verification verb exit, which sends a request to the partner LU to remove the entry from the signed-on-to list. When the request is complete, the operator receives RACF message IRRE006I, which contains a return and reason code. The return and reason codes are documented in [z/OS Security Server RACROUTE Macro Reference](#). The return and reason codes for the persistent verification verb exit are:

Table 12. Persistent Verification Verb Exit Return and Reason Codes		
Return Code	Reason Code	Meaning and Action
Hexadecimal (Decimal)	Hexadecimal (Decimal)	Meaning
00 (00)	--	Processing completed successfully.
0C (12)	04 (04)	The request code passed to the persistent verification verb exit was not correct.
	08 (08)	APPC/MVS is not active.
	0C (12)	An internal error occurred.
	10 (16)	An internal error occurred.

If you are using a security product other than RACF, the persistent verification verb exit return and reason codes may be different from those listed above.

## Diagnosing PV Problems

There are some situations in which signed-on-to and signed-on-from lists might not be identical, possibly causing an erroneous return code of security\_not\_valid from an allocate request. For example, a partner application might have two windows, one that starts a conversation with a "sign-on" request, and a second that starts another conversation to the same LU and passes a "signed-on" indicator. If for some

reason the second allocate reached the inbound LU first, the user ID would not be found on the signed-on-from list yet, and the "signed-on" request would be rejected. The user would need to try that request again later.

See *SNA LU 6.2 Reference: Peer Protocols* for more information about using persistent verification.

End Programming Interface Information

## Diagnosing Security Problems

When an outbound TP's allocate request includes inadequate security information to access its partner LU or TP on MVS, APPC/MVS passes a return code of *security\_not\_valid* on a call following the allocate call. For more information about that return code, see *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*. RACF messages on the inbound system indicate any errors in the specification of RACF profile information. APPC/MVS messages indicate errors in specifying security information in TP profiles and side information. For explanations of the diagnostic messages, see *z/OS Security Server RACF Messages and Codes* and *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

As an aid to debugging access problems and detecting security exposures, you can use RACF auditing to keep track of access to RACF profiles. For example, you can audit all attempted accesses, all successful accesses, and all failed accesses to a particular profile or class of RACF profiles.

You can also use APPC component trace options to collect and diagnose the security information that crosses the network in an allocate request. See *z/OS MVS Diagnosis: Tools and Service Aids* for information about requesting and formatting FMH-5 data with APPC component trace.

Programming Interface Information

## Maintaining MVS Passwords in an APPC Environment

The passwords associated with RACF user profiles periodically expire and must be updated by the user. An installation that uses APPC to communicate with TPs running on MVS may want to use APPC to update MVS passwords.

APPC/MVS provides a TP (X'06F3F0F1') that makes it possible to update RACF passwords on MVS, and two sample TPs (ATBMINO and COMUPASS) that aid in managing passwords:

### **X'06F3F0F1'**

SIGNON/Change password (SNA name X'06F3F0F1') is an internal implementation of an SNA service transaction program that maintains passwords on APPC/MVS. The program is invoked through an APPC attach. For more information, see [“What is the SIGNON/Change Password TP?”](#) on page 160.

### **ATBMINO**

The source code for ATBMINO is a sample implementation of SNA program X'30F0F5F2'. The expired password notification TP (SNA name X'30F0F5F2') has not been implemented in APPC/MVS. The sample implementation of this program (ATBMINO) runs on a workstation. A request to attach X'30F0F5F2' indicates expiration of the password for the user ID specified in an FMH-5 passed from the partner LU running APPC/MVS. If an installation chooses to implement a version of this TP on an z/OS system, APPC/MVS will allow the TP to run with an expired password. The source code for ATBMINO is located in SYS1.SAMPLIB(ATBMINOS). The executable code for ATBMINO is located in SYS1.SAMPLIB(ATBMINO).

### **COMUPASS**

This is an example of a program that allows APPC users to specify a new MVS password. The source code for the sample implementation of the program is located in SYS1.SAMPLIB(ATBMIPWS). The executable code is located in SYS1.SAMPLIB(COMUPASS).

The ATBMINO and COMUPASS password maintenance TPs comprise *examples* of how a partner LU can communicate with APPC/MVS to maintain passwords. These examples are shipped with MVS, and may be downloaded to the system running on the partner LU. For instructions on how to download and use these programs, see [“Using Sample Programs to Maintain User Passwords on a Partner LU”](#) on page 165.

As an alternative to modifying the sample programs to maintain APPC/MVS passwords, partner systems can use the SIGNON/Change password TP as described in [“2.c” on page 119](#). This can avoid the rejection of an allocate (attach) request due to an expired user password.

## What is the SIGNON/Change Password TP?

The SIGNON/Change password SNA service TP (SNA name X'06F3F0F1') runs on APPC/MVS and does the following:

- Signs on users to a server LU to support LU 6.2 persistent verification (PV).

A requestor LU can invoke this function on behalf of a user to ensure that the user's password has not expired before sending an attach request to a server LU. If the SIGNON/Change password TP indicates that the password is expired, the LU can prompt the user for a new password and invoke the SIGNON/Change password TP again to change the user's MVS password.

After the SIGNON/Change password TP confirms that the user is signed on with an unexpired password, and the requestor LU receives a generalized data stream (GDS) variable indicating that the user is signed on, the LU can add the user to the signed-on-to list and send the user's original attach request. For more information on the SIGNON/Change password GDS variable, see [“Description of SIGNON/Change Password GDS Variable” on page 161](#).

This process may be implemented with an application TP, or it may be used in an implementation of PV for an LU. With PV, SIGNON/Change password should be invoked only once for all of a user's conversations in a session. The sending LU may check its own signed-on-to list and invoke the SIGNON/Change password TP if the user is not in the list.

- Signs on users to a server LU and changes user passwords on the server LU.

## How to Create Partner LU Communication for the SIGNON/Change Password TP

Figure 94 on [page 160](#) shows an example of how a partner LU can use the SIGNON/Change password TP to verify that a user's password has not expired on APPC/MVS before flowing an attach request:

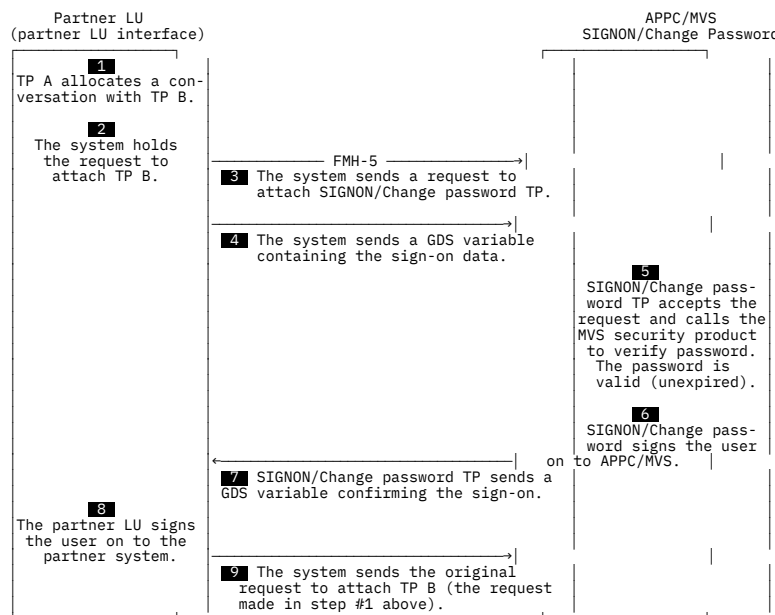


Figure 94. Signing a User On to APPC/MVS (Unexpired Password)

Figure 95 on [page 161](#) shows an example of how a partner LU can use the SIGNON/Change password TP to prevent rejection of an attach request due to the expiration of a user's password:

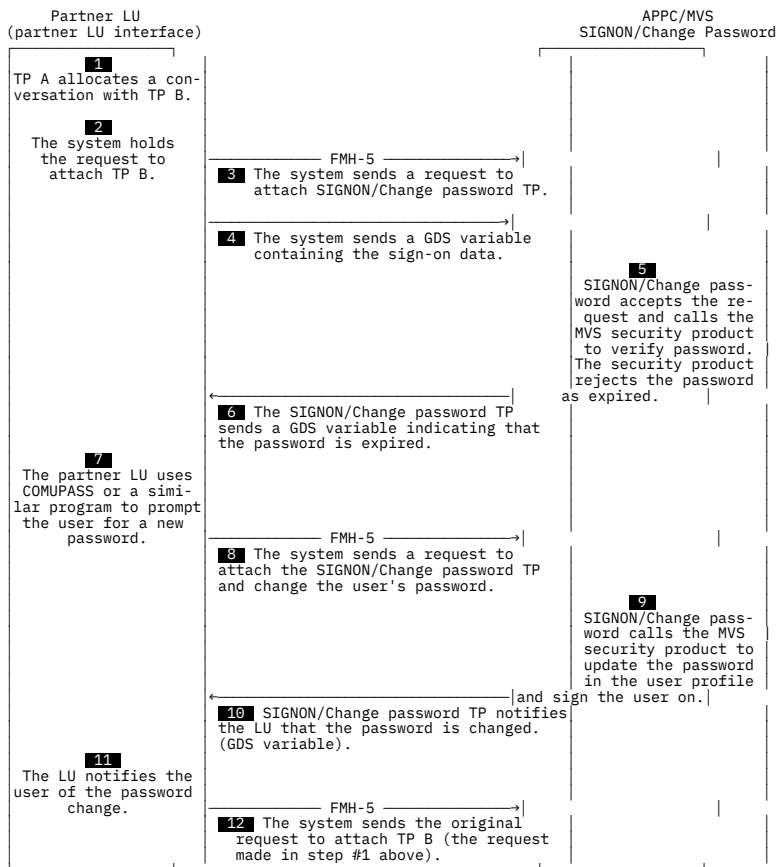


Figure 95. Signing a User On to APPC/MVS (Changing an Expired Password)

**Note:** APPC/MVS rejects the attach request if the SIGNON/Change password TP is invoked with a conversation security level other than security (none).

When creating a partner system interface to the SIGNON/Change password TP for an installation, the system programmer should follow these guidelines:

- In multi-tasking systems, where it is possible for more than one TP to start on parallel sessions, the code that handles user allocate requests should also serialize the process of signing users on to the system. This causes an allocate for a second process to wait for completion of a SIGNON initiated for the first process.
- When a user issues an allocate request after a PV sign-off, the LU should serialize the sign-on process and sign the user on to the system again.
- The mode name for the SIGNON/Change password TP should be the same as the mode name specified on the user allocate request.
- If PV is used, the signed-on-to list must be accessible from the LU that requests the user sign-on.

## Description of SIGNON/Change Password GDS Variable

The SIGNON/Change password GDS variable contains information needed for a requestor LU to send a *sign-on* request or *sign-on with a password change* request to a server LU, or information needed for a server LU to send a reply to those requests.

### General GDS Variable Format

Table 13 on page 162 shows the general format of the header of the SIGNON/Change password GDS variable. The remaining tables in this section show the sub-fields that are required on the SIGNON/Change password GDS variable for each type of request or reply.



Table 13. General Format of SIGNON/Change Password GDS Variable		
Byte(s)	Value	Description of Contents
0-1	xxxx	xxxx - The length of the entire GDS variable, including the length of this field.
2-3	X'1221'	The SIGNON/Change password GDS variable identifier.
4-5	yyyy	yyyy is the length of this field, plus the length of all the fields that follow this field in the GDS variable (the length of the nested data structure).
6-7	zzzz	zzzz is one of the following: <b>X'FF00'</b> The request is to sign a user on. The allocate request flows from the requestor LU to the server LU. See <a href="#">Table 14 on page 162</a> for the sub-fields that accompany this request. <b>X'FF01'</b> The request is to sign the user on and change the user's password. The allocate request flows from the requestor LU to the server LU. See <a href="#">Table 15 on page 162</a> for the sub-fields that accompany this request. <b>X'FF02'</b> The server LU sends a reply to the requestor LU that contains the results of a "sign-on" or "sign-on and change password" request. See <a href="#">Table 16 on page 163</a> for the sub-fields that accompany this request.

### Format for Sign-on (FF00) Sub-fields

The following sub-fields follow byte 5 in [Table 13 on page 162](#) when a partner LU sends a request is to sign a user on (FF00):

Table 14. Format of GDS Variable Sub-fields For A Sign-on Request		
Field Length	Identifier	Description of Contents
3-12	X'00'	User security profile name (optional)
3-12	X'01'	User ID
3-12	X'02'	Password

### Format for SIGNON/Change Password (FF01) Sub-fields

The following sub-fields follow byte 5 in [Table 13 on page 162](#) when a partner LU sends a request to sign a user on and change the user's password (FF01):

Table 15. Format of GDS Variable Sub-fields For A Sign-on and Change Password Request		
Field Length	Identifier	Description of Contents
3-12	X'00'	User security profile name (optional)
3-12	X'01'	User ID
3-12	X'02'	Old (expired) password
3-12	X'06'	New password



### Format for SIGNON Reply Data (FF02) Sub-fields

The following sub-fields follow byte 5 in [Table 13 on page 162](#) when the SIGNON/Change password TP sends a reply to a request from the partner LU (FF02):

Table 16. Format of GDS Variable Sub-fields For SIGNON/Change Password Reply		
Field Length	Identifier	Description of Contents
3	X'00'	SIGNON/Change password status value (see “SIGNON/Change Password Status Values” on page 164)
4	X'01'	SIGNON request formatting error; this sub-field is present only with status value 6

### Example GDS Variable - Input to SIGNON/Change Password TP

The following is an example of a GDS variable that a system programmer can create as *input* to the SIGNON/Change password TP. In the figure, sub-field blocks 5-7, 8-10, 11-13, and 14-16 can appear in any order. Sub-field block 5-7 is optional.

Table 17. Example GDS Variable (SIGNON/Change password TP input)			
Field Number	Length Byte(s)	Value	Description of Contents
1	2	xxxx	xxxx - The length of the entire GDS variable, including the length of this field.
2	2	X'1221'	The SIGNON/Change password GDS variable identifier.
3	2	xxxx	Length of field 3, plus the length of all the fields that follow this field in the GDS variable (length of nested data structure)
4	2	FF01	Request identifier indicating a request to sign a user on and change the user's password
5	1	xx	Length of fields 5-7
6	1	X'00'	Identifier for profile field
7	x	C'profname'	profname - user profile name
8	1	xx	Length of fields 8-10
9	1	X'01'	Identifier for user ID
10	x	C'userid'	userid - User ID for user issuing the allocate request
11	1	xx	Length of fields 11-13
12	1	X'02'	Identifier for old (expired) password
13	x	C'oldpass'	oldpass - User's old (expired) password
14	1	xx	Length of fields 14-16
15	1	X'06'	Identifier for new password (required field, only allowed with request identifier FF01)
16	x	C'newpass'	newpass - User's new password

## GDS Variable - Output from SIGNON/Change Password TP

The following figure shows the GDS variable that APPC/MVS returns to the partner LU to indicate the response of the SIGNON/Change password TP. Fields 8-10 are present only when the status value in field 7 is X'06'.

Table 18. GDS Variable Structure (SIGNON/Change password TP output)			
Field Number	Length Byte(s)	Value	Description of Contents
1	2	xxxx	xxxx - The length of the entire GDS variable, including the length of this field.
2	2	X'1221'	The SIGNON/Change password GDS variable identifier.
3	2	xxxx	Length of field 3, plus the length of all the fields that follow this field in the GDS variable (length of nested data structure)
4	2	FF02	SIGNON/Change password reply identifier
5	1	xx	Length of fields 5-7
6	1	X'00'	SIGNON/Change password status value identifier
7	1	xx	SIGNON/Change password status value (see <a href="#">“SIGNON/Change Password Status Values”</a> on page 164 for a list of status values)
8	1	xx	Length of fields 8-10
9	1	X'01'	SIGNON/Change password request formatting error identifier
10	2	xxxx	SIGNON/Change password request formatting error value described in <a href="#">“SIGNON/Change Password Request Formatting Errors”</a> on page 165 (this value only appears when status value 6 appears in field 7)

## SIGNON/Change Password Status Values

Table 19 on page 164 lists the password status values that the SIGNON/Change password TP returns in the GDS variable, to indicate the status of a password change request to the partner system. For additional diagnostic information, the table also includes the return code that APPC/MVS receives from the security product.

Table 19. SIGNON/Change Password TP Status Values		
Status Value	Description	Return Code
00	APPC/MVS successfully processed the request. The user is signed on. If a password change was requested, the password is changed.	00
01	The user ID is not known to APPC/MVS. The user profile is not defined to the security product on MVS.	04
02	The user ID is valid, but the password is incorrect.	08
03	The old password is correct but it has expired.	0C
04	The user ID is valid and the password is correct, but the new password is not acceptable to the security product on MVS.	10

Table 19. SIGNON/Change Password TP Status Values (continued)		
Status Value	Description	Return Code
05	This value is expected when one of the following is true: <ul style="list-style-type: none"> <li>The security product on MVS is either not installed or not active</li> <li>An error occurred in the SIGNON/Change password TP</li> <li>For X'FF00' requests only, the security product on MVS is not set up correctly for persistent verification.</li> </ul>	Undetermined
06	Incorrect data format. The specific error is contained in the SIGNON/Change password TP request formatting error field described in <a href="#">“SIGNON/Change Password Request Formatting Errors”</a> on page 165.	N/A
07	General security error.	Other
08	The SIGNON/Change password TP successfully completed a password change, but could not add the user to the signed-on-from list.  If you are <b>not</b> using persistent verification, receiving this status value is equivalent to receiving a status value of 00; no corrective action is necessary.  Otherwise, status value 08 is expected when the security product on MVS is not set up correctly for persistent verification. Refer to the security product documentation to determine what must be corrected. If your installation is using RACF, the RACF subsystem must be available to handle persistent verification requests. Refer to <a href="#">z/OS Security Server RACF System Programmer's Guide</a> for more information about the RACF subsystem.	Undetermined

## SIGNON/Change Password Request Formatting Errors

The SIGNON/Change password TP returns one of the following request formatting errors in the GDS variable (see [Table 16](#) on page 163) when a status value of 6 appears in field 7 of the GDS variable:

Table 20. SIGNON/Change Password Request Formatting Errors	
Format Error	Description
0000	Undefined error
0001	Required structure absent
0002	Precluded structure present
0003	Multiple occurrences of a non-repeatable structure
0006	Length outside specified range
0007	Length exception - length arithmetic is out of balance
000C	Required combination of structures is absent

For more information on how to create a partner system interface to the SIGNON/Change password TP, see *SNA Formats* and *SNA LU 6.2 Reference: Peer Protocols*

## Using Sample Programs to Maintain User Passwords on a Partner LU

IBM provides two sample programs that can help you create a password management system for end users of APPC/MVS applications. The sample programs, called ATBMINO and COMUPASS, enable end users of APPC/MVS applications to update passwords on a partner LU running APPC/MVS. Specifically, the sample programs do the following:

- Display a panel on a remote workstation to notify a user of an expired APPC/MVS password
- Allow a user to enter a new password
- Send a request to update a password in the MVS security product user profile.

Without the sample programs, APPC/MVS users must access MVS through an environment such as an LU2 Time Sharing Option Extensions (TSO/E) session to change a password on a partner LU running

APPC/MVS. If the user does not update the password on both systems, APPC/MVS rejects further Allocate (attach) requests from the user.

With password maintenance sample programs like those shipped with MVS/ESA, APPC/MVS users no longer need to be defined to an interactive LU2 host environment, and may not need an LU2 terminal emulation connection if they used it only to change their password.

To use the password maintenance sample programs, the RACF component of the z/OS Security Server or an equivalent security product must be installed. The sample programs were designed to run on OS/2, but you can tailor them to work on any operation system.

**Note:** The sample programs *do not* update passwords in the workstation's local security database.

## ATBMINO and COMUPASS Sample Programs

To maintain passwords remotely, you may install the ATBMINO and COMUPASS sample programs on your platform. While these samples are designed to run on an OS/2 workstation, they can be modified to meet the needs of any platform or installation. For information on how to install the sample programs, see [“How to Install the Sample Programs that Maintain Passwords”](#) on page 178.

The ATBMINO and COMUPASS programs are described below:

### ATBMINO

A SNA service transaction program (TP) that does the following:

- Accepts notification of an expired MVS password from a server LU running APPC/MVS
- Notifies the user of an expired password by displaying a pop-up panel on the workstation.
- Invokes the COMUPASS sample program, if the user decides to change an expired password immediately.

The ATBMINO sample program is an implementation of SNA service TP X'30F0F5F2'.

### COMUPASS

A TP that does the following:

- Prompts the user for information needed to change the user's password in the security product database on APPC/MVS
- Requests that the partner LU change the password in its security database.

Users can use the COMUPASS program to update an APPC/MVS password at any time, whether the password has expired or not.

## A Typical Scenario - Changing an Expired Password

Figure 96 on page 167 shows a typical scenario in which an expired APPC/MVS password is updated. In the figure, the ATBMINO sample program is an implementation of the expired password notification SNA service TP X'30F0F5F2'.

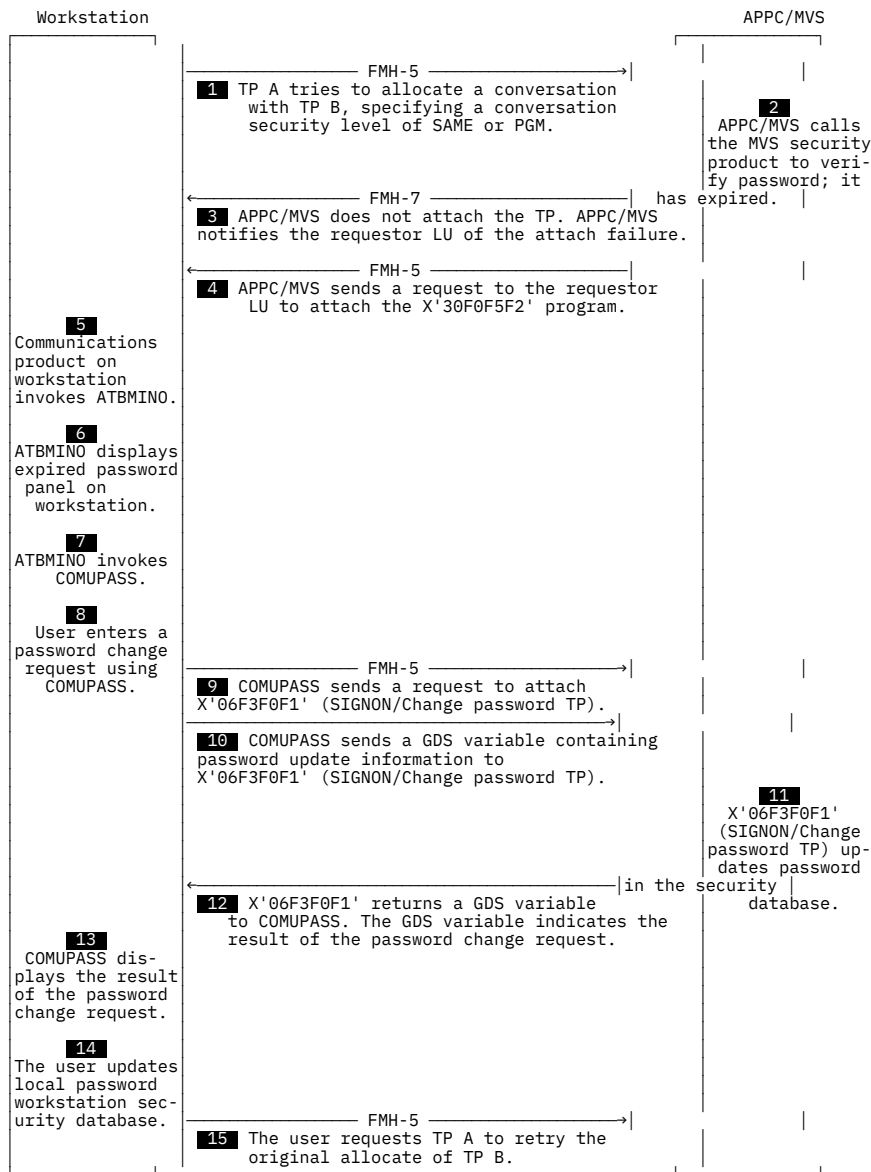


Figure 96. Changing an Expired Password

## Changing a Password that Has Not Expired

To change an APPC/MVS password that has not yet expired, a user can enter the COMUPASS command on the command line using the following syntax:

```
COMUPASS userid plualias mode_name old_password new_password
```

In the command text:

### **userid**

The userid associated with the password to be changed.

### **plualias**

The partner LU alias for the system on which the password is to be changed.

### **mode\_name**

The transmission service mode name for the partner LU.

### **old\_password**

The expired partner LU password.

### ***new\_password***

The new partner LU password. The maximum length of passwords that COMUPASS will accept is ten. However, the password length must be acceptable to the partner LU security product. APPC/MVS limits the password length to eight characters.

The COMUPASS program prompts the user for parameters that are not supplied on the command invocation. Because the parameters are positional, if the user omits a parameter from the command invocation, all parameters that follow the omitted parameter must also be omitted from the command invocation.

## **Diagnosing Problems when Using the Password Maintenance Sample Programs**

This section shows you how to diagnose errors that occur when running the ATBMINO and COMUPASS programs. Users receive error messages from APPC/MVS in the following forms:

- ATBMINO error panels
- COMUPASS error panels
- COMUPASS user messages.

### **ATBMINO Error Panels**

The ATBMINO program displays the following panels on the OS/2 workstation when ATBMINO cannot notify the user of an expired password:

<p>In attempting to notify you of an expired password on a partner logical unit, program <b>ATBMINO</b> has issued the APPC verb which failed.</p> <p>Contact installation support personnel with the following information:</p> <p>Failed APPC verb name : <b>verb name</b> Primary return code : <b>primary rc</b> Secondary return code : <b>secondary rc</b></p>
<p>Press ESCAPE (ESC) to remove this window.</p>

*Figure 97. APPC Error Notification Panel*

This error message indicates that a failure occurred in the conversation with the partner LU. For descriptions of the primary and secondary return codes, see the communication product's APPC documentation.

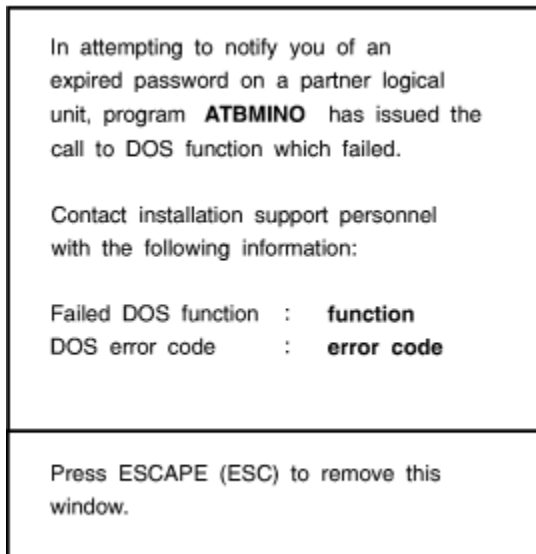
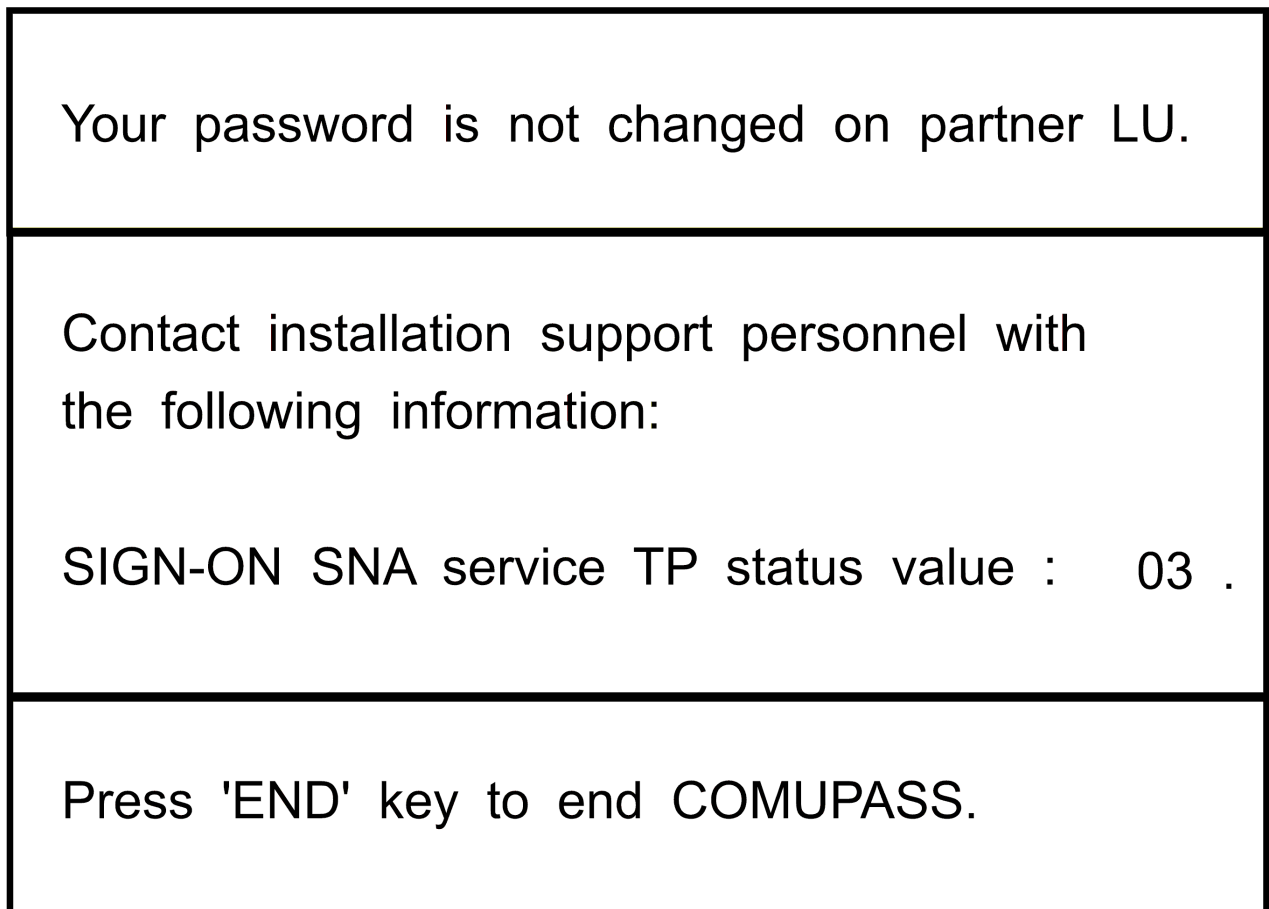


Figure 98. DOS Error Notification Panel

This error message indicates that a failure occurred in a call to DOS. For descriptions of possible DOS error codes, see *PC DOS Command Reference and Error Messages*.

### COMUPASS Error Panels

The COMUPASS program displays panels on the workstation when an error occurs during password-change processing. The text following each panel explains and provides responses for the error condition.



If COMUPASS receives status value 03 from the SIGNON/Change password TP on the partner LU, it is likely that COMUPASS was incorrectly modified. Contact your system administrator, who can diagnose the problem using [Table 19 on page 164](#).

Your password is not changed on partner LU.

Contact installation support personnel with the following information:

SIGN-ON SNA service TP status value : 03 .

Press 'END' key to end COMUPASS.

COMUPASS receives this SIGN-ON SNA service TP status value when the partner LU security product is not installed or not active. Contact your system administrator, who can diagnose the problem using [Table 19 on page 164](#).



Your password is not changed.

General security failure on partner LU.

Contact installation support personnel with the following information:

SIGN-ON SNA service TP status value: 07 .

Press 'END' key to end COMUPASS.

COMUPASS receives this SIGN-ON SNA service TP status value when an error occurred in the MVS security product. In this case, specify a valid userid, or password, or both. If the error occurs again, contact your system administrator, who can diagnose the problem using [Table 19 on page 164](#).

Your password is not changed.

Incorrect data format in Generalized Data Stream (GDS) variable sent to partner LU.

Contact installation support personnel with the following information:

SIGN-ON SNA service TP format error  
code : **format\_error.**

Press 'END' key to end COMUPASS.

This panel indicates that the format of the GDS variable, which COMUPASS uses to pass information about a password change request to the partner LU, was changed in the COMUPASS program. The format is no longer acceptable to the MVS security product.

In the panel text:

***format\_error***

The SIGN-ON service TP format error code. For an explanation of the code, see the communication product's APPC documentation.

Contact the system programmer to change the code in the COMUPASS program so it passes a GDS variable with the correct format. For the correct GDS variable format, see [Table 13 on page 162](#).

Your password may not be changed.
<p>Unrecognized status value is received from partner LU.</p> <p>Contact installation support personnel with the following information:</p> <p>SIGN-ON SNA service TP status value : <b>xx</b>.</p>
Press 'END' key to end COMUPASS.

COMUPASS received an unexpected status value from the SIGN-ON SNA service TP. Provide that status value to the APPC/z/OS system administrator, who can diagnose the problem using [Table 19 on page 164](#).

Your password was not changed.
<p>Contact installation support personnel with the following information:</p> <p>Failed APPC verb name : <b>verb name</b></p> <p>PLU Alias : <b>alias</b></p> <p>Primary return code : <b>primary rc</b></p> <p>Secondary return code : <b>secondary rc</b></p>
Press 'END' key to end COMUPASS.

This panel indicates that a failure occurred while setting up, allocating, or sending data in the conversation with the partner LU. For descriptions of the verbs and primary and secondary return codes, see the communication product's APPC documentation.

Your password may not be changed.

Contact installation support personnel with the following information:

Failed APPC verb name : RECV\_AND\_WAIT

PLU Alias : **alias**

Primary return code : **primary rc**

Secondary return code : **secondary rc**

Press 'END' key to end COMUPASS.

This panel indicates that a failure occurred while COMUPASS was waiting to receive data from the partner LU. For descriptions of the verb and primary and secondary return codes, see the communication product's APPC documentation.

Contact installation support personnel with the following information:

Failed APPC verb name : RECV\_AND\_WAIT

PLU Alias : **alias**

Primary return code : **primary rc**

Secondary return code : **secondary rc**

Press 'END' key to end COMUPASS.

This panel also indicates that a failure occurred while COMUPASS was waiting to receive data from the partner LU. For descriptions of the verb and primary and secondary return codes, see the communication product's APPC documentation.

Contact installation support personnel with the following information:	
Failed APPC verb name	: TP_ENDED
PLU Alias	: <b>alias</b>
Primary return code	: <b>primary rc</b>
Secondary return code	: <b>secondary rc</b>
Press 'END' key to end COMUPASS.	

This panel indicates that a failure occurred while COMUPASS was ending its conversation with the partner LU. For descriptions of the verb and primary and secondary return codes, see the communication product's APPC documentation.

## COMUPASS Messages

The COMUPASS program displays messages on the workstation to indicate the status of a password change request. Most of the following COMUPASS messages display information about user errors, but some may require system administrator intervention.

The following are the COMUPASS messages, listed in alphabetical order by message text. (The messages are not issued with traditional message ID numbers but, for clarity, are identified here as "Message 1," "Message 2," and so on.)

---

**Message 1**      **Enter valid userid for partner LU or press ESCAPE (ESC) to terminate password update:**

---

### Explanation

One of the following occurred when the user tried to run COMUPASS:

- The user did not specify a userid.
- The user specified a userid that was not valid.

### User response

Enter a valid userid. Press ENTER.

---

**Message 2**      **Enter valid partner LU alias or press ESCAPE (ESC) to terminate password update:**

---

### Explanation

One of the following occurred when the user tried to run COMUPASS:

- The user did not specify a partner LU alias.
- The user specified a partner LU alias that was not valid.

### User response

Enter a valid partner LU alias. Press ENTER.

---

**Message 3**      **Enter valid mode name or press ESCAPE (ESC) to terminate password update:**

---

### Explanation

One of the following occurred when the user tried to run COMUPASS:

- The user did not specify a mode name.
- The user specified a mode name that was not valid.

### User response

Enter a valid mode name. Press ENTER.

---

**Message 4**      **Enter valid old password for partner LU or press ESCAPE (ESC) to terminate password update:**

---

### Explanation

One of the following occurred when the user tried to run COMUPASS:

- The user did not specify an old password.
- The user specified an old password that was not valid.

### User response

Enter a valid old password. Press ENTER.

---

**Message 5**      **Enter valid new password for partner LU or press ESCAPE (ESC) to terminate password update:**

---

### Explanation

One of the following occurred when the user tried to run COMUPASS:

- The user did not specify a new password.
- The user specified a new password that was not valid.

### User response

Enter a valid new password. Press ENTER. When the COMUPASS prompts you with **Enter new password for partner LU again;**, enter the same password again to verify it.

---

**Message 6**      **You did not specify the same new password both times.**

---

### Explanation

A new password does not match the previously entered new password.

### User response

Enter a valid new password. Press ENTER.

---

**Message 7**      **The supplied userid for partner LU is incorrect.**

---

### Explanation

When trying to run COMUPASS, the user specified a userid that is not valid on the partner LU.

## System action

The COMUPASS program prompts the user to enter a valid userid.

## User response

Enter a userid with a character length that is valid on the partner LU. Press ENTER.

---

<b>Message 8</b>	<b>The supplied partner LU alias is incorrect.</b>
------------------	--

---

## Explanation

When trying to run COMUPASS, the user specified a partner LU alias that is not valid on the partner LU.

## System action

The COMUPASS program prompts the user to enter a valid partner LU alias.

## User response

Enter a partner LU alias with a character length that is valid on the partner LU. Press ENTER.

---

<b>Message 9</b>	<b>The supplied mode name is incorrect.</b>
------------------	---

---

## Explanation

When trying to run COMUPASS, the user specified a mode name that is not valid on the partner LU.

## System action

The COMUPASS program prompts the user to enter a valid mode name.

## User response

Enter a mode name with a character length that is valid on the partner LU. Press ENTER.

---

<b>Message 10</b>	<b>The supplied old password for partner LU is incorrect.</b>
-------------------	---

---

## Explanation

When trying to run COMUPASS, the user specified an old password that was not valid on the partner LU.

## System action

The COMUPASS program prompts the user to enter a valid old password.

## User response

Enter an old password with a character length that is valid on the partner LU. Press ENTER.

---

<b>Message 11</b>	<b>The supplied new password for partner LU is incorrect.</b>
-------------------	---

---

## Explanation

When trying to run COMUPASS, the user specified a new password that is not valid on the partner LU.

## System action

The COMUPASS program prompts the user to enter a valid new password.

## User response

Enter a new password with a character length that is valid on the partner LU. Press ENTER.

---

<b>Message 12</b>	<b>Your password is not changed. Mode name or partner LU alias is not valid.</b>
-------------------	--

---

## Explanation

APPC/MVS rejected one of the following:

- The mode name.
- The partner LU alias.

## System action

The COMUPASS program prompts the user to enter a valid mode name or partner LU alias.

## User response

Enter a valid partner LU alias or mode name. Press ENTER.

---

<b>Message 13</b>	<b>Password change completed on partner LU LU alias Press 'END' key to end COMUPASS. NOTE : Please also update your local security data (APPC Conversation Security and/or UPM).</b>
-------------------	--

---

## Explanation

SIGNON/Change password successfully completed a password change.

In the message text:

### **LU alias**

The partner LU alias.

### User response

Press the END key to end COMUPASS. Update the local security database with the new password.

---

**Message 14      The supplied userid is not valid on partner LU.**

### Explanation

The userid is not defined to the MVS security product.

### System action

The COMUPASS program prompts the user to enter a valid userid for the partner LU.

### User response

Enter a valid userid. Press ENTER. If the error message continues to appear, contact your system administrator.

### Programmer response

Contact the MVS security administrator to define the user profile to the MVS security manager.

---

**Message 15      The supplied old password is not valid on partner LU.**

### Explanation

When trying to run COMUPASS, the user specified an old password that is not valid on the partner LU.

### System action

The COMUPASS program prompts the user to enter a valid old password.

### User response

Enter a valid old password. Press ENTER.

---

**Message 16      The supplied new password is not valid on partner LU.**

### Explanation

When trying to run COMUPASS, the user specified a new password that is not acceptable to the MVS security product.

### System action

The COMUPASS program prompts the user to enter a valid new password.

### User response

Enter a valid new password. Press ENTER.

## How to Install the Sample Programs that Maintain Passwords

This chapter describes how to install the ATBMINO and COMUPASS sample programs.

### MVS Data Sets

IBM provides MVS data sets that contain both the source and executable code for the sample programs. Download the **source code** for the sample programs if you wish to modify them to meet the needs of your installation. Download the **executable code** for the sample programs if you wish to run the programs without modifying them.

The **source code** for the sample programs is contained in the following data set members:

SYS1.SAMPLIB(ATBMINOS)  
SYS1.SAMPLIB(ATBMIPWS)

The **executable code** for the sample programs is contained in the following data set members:

SYS1.SAMPLIB(ATBMINO)  
SYS1.SAMPLIB(COMUPASS)

### Installation Procedure for Source Code

Install the sample programs (source code) using the following procedure:



Task	Reference
Compile the programs using the C compiler with the appropriate include files.	<a href="#">“Compiling the Sample Programs (Source Code Only)” on page 179</a>
Define the ATBMINO program to the APPC communications product as a remotely attachable TP.	<a href="#">“Defining the ATBMINO Program to APPC on the Workstation” on page 179</a>
Create a conversation security profile, or define an entry in the security product, for the user.	<a href="#">“Defining Conversation Security” on page 180</a>

## Installation Procedure for Executable Code

Install the sample programs (executable code) using the following procedure:

Task	Reference
Download the executable code from APPC/MVS.	<a href="#">“Downloading the Sample Programs (Executable Code)” on page 179</a>
Define the ATBMINO program the APPC communications product as a remotely attachable TP.	<a href="#">“Defining the ATBMINO Program to APPC on the Workstation” on page 179</a>
Create a conversation security profile, or define an entry in the security product, for the user.	<a href="#">“Defining Conversation Security” on page 180</a>

## Downloading the Sample Programs (Executable Code)

If you are using OS/2 you can download the executable code for the sample programs as is, without any modifications. When doing this, ensure that the COMUPASS program is placed in a directory that is currently defined in the CONFIG.SYS file on the OS/2 workstation.

## Compiling the Sample Programs (Source Code Only)

Because these sample programs were originally written in IBM C/2 for OS/2, you must modify all of the OS/2-specific invocations so that they run on the particular platform you are supporting. If you are compiling on OS/2, the sample programs will probably compile with little or no modifications.

In addition, you will need to make sure that the necessary APPC data structures that the program accesses are available on your system. See the APPC programming documentation for the platform you are writing this application on for further details.

When linking the program, you will probably need to link the appropriate APPC run-time libraries with the sample programs. See the APPC programming documentation for the platform you are writing this application on for further details.

## Defining the ATBMINO Program to APPC on the Workstation

This section describes how to define the ATBMINO password expiration notification program to the workstation. Perform this procedure only if the user requires password expiration notification.

1. Go to the TP definition section of the APPC communications product you are using.
2. Specify the TP Profile name as a SNA service TP name of X'30',052. See the APPC configuration documentation for further details on SNA Service TP profile names.
3. If the APPC communications product allows you to specify a Sync Level, specify None.
4. If the APPC communications product allows you to specify a Conversation Type, specify Basic.
5. If the APPC communications product allows you to specify a Conversation Security value, specify Yes.

Again, see the APPC configuration documentation for the workstation you are installing this TP on for more details on defining TPs on

## Defining Conversation Security

Most APPC communication products allow you to define a list of userids and passwords that are valid for incoming allocation requests, or allow you to consult the security product for the password validation. See the APPC configuration documentation on your platform for further details.

End Programming Interface Information

## Encrypting Data and Security Information

To ensure the security and integrity of all application information that crosses the network, your installation can use a product that allows encryption and decryption of data and security information on the systems on which conversing TPs run. One such product for use on z/OS systems is the Integrated Cryptographic Service Facility/MVS (ICSF/MVS). With ICSF/MVS, your installation can either:

- Design conversing TPs to use the high-level language callable service interface of ICSF/MVS to encrypt data before sending and decrypt data after receiving it through a conversation. APPC/MVS will send and receive encrypted data and security information without affecting it.
- Use VTAM to encrypt and decrypt all data that flows across the session, with no action required by the conversing TPs. For more information about using VTAM to encrypt and decrypt data, see [z/OS Communications Server: SNA Programming](#).

If you use a program such as ICSF/MVS, you need to have compatible cryptographic products available on the non-MVS partner systems for the partner TPs to use. For details on setting up ICSF/MVS on an z/OS system, see [z/OS Cryptographic Services ICSF System Programmer's Guide](#).

## Summary

APPC/MVS and RACF provide a number of security mechanisms to protect APPC/MVS TPs and the z/OS system in a cooperative processing environment. The mechanisms that you use depend on the APPC applications you install and the extent to which you want to protect them.

APPC applications may provide for security by passing security information, including user IDs and passwords, on allocate requests from the outbound to the inbound TP. Table 21 on page 180 shows what LU or conversation security mechanisms you can implement, based on the security information passed.

Table 21. Security Mechanisms Available, based on Application Security Types		
Application Security	LU Security Mechanisms	Conversation Security Mechanisms
NONE	Supported	Not supported
SAME	Supported	Supported
PGM	Supported	Supported

You can use the LU security mechanisms for any application, including those that pass no security information. The LU security mechanisms protect APPC/MVS logical units, by verifying the authority of other LUs to establish communication sessions with them, and the level of security needed on any conversation that crosses the session. You can also ensure that specific LU names are defined to VTAM by APPC only.

You can use the conversation security mechanisms for applications that pass a user ID on the allocate request. The conversation security mechanisms provide a security environment for TPs running on MVS, and they let you control access, by user ID, to LUs, from LUs, and to TPs and related information.

Finally, for higher security in a cooperative processing environment, you can use a cryptographic product such as IBM's Integrated Cryptographic Service Facility/MVS to encrypt and decrypt data and security information that crosses the network between TPs in an APPC application.

Table 22 on page 181 summarizes RACF classes and resource names, and how they affect APPC.

<i>Table 22. Summary of RACF Classes and Resource Names</i>	
<b>RACF class and profile naming convention</b>	<b>Function</b>
In class APPCLU, profiles with the following names: lnetwork-id.local-lu-name.pnetwork-id.partner-lu-name lnetwork-id.local-lu-name.partner-lu-name lnetwork-id.generic-name.pnetwork-id.partner-lu-name lnetwork-id.generic-name.partner-lu-name	Define LU-LU security.  See <a href="#">“Defining LU-to-LU Access Authority with RACF APPCLU Profiles” on page 139</a>
In class APPCPORT, profiles with the following names: luname	Control which LU the user's request can come from.  See <a href="#">“Controlling User Access from LUs” on page 148</a>
In class APPCSERV, profiles with the following names: dbtoken.tpname	Control server access to clients.  See <a href="#">z/OS MVS Programming: Writing Servers for APPC/MVS</a> .
In class APPCSI, profiles with the following names: dbtoken.SYS1.symdname	Control the user's access to side information.  See <a href="#">“Controlling User Access to Side Information” on page 150</a>
In class APPCTP, profiles with the following names: dbtoken.level.tpname	Control the user's access to TP profiles.  See <a href="#">“Controlling User Access to TPs” on page 151</a>
In class APPL, profiles with the following names: luname  or genericname	Control use of local LUs.  See <a href="#">“Controlling User Access to LUs” on page 145</a>
In class FACILITY, a profile with the following name: APPCMVS.DBTOKEN	Controls the user's access to database tokens.  See <a href="#">“Controlling Access to Database Tokens” on page 150</a>
In class FACILITY, profiles with the following names: APPCMVS.TP.MULTI.generic-userid	Control the user's access to multi-trans TP profiles.  See <a href="#">“Protecting Multi-Trans TP Profiles” on page 153</a>
In class FACILITY, profiles with the following names: ATBTRACE.netid.lu_name.tp_name	Controls the user's ability to collect API trace data.  See <a href="#">“Controlling Ability to Collect API Trace Data” on page 154</a>

Table 22. Summary of RACF Classes and Resource Names (continued)

RACF class and profile naming convention	Function
<p>In class PROGRAM, profiles with the following names:</p> <p>ATBINMIG</p> <p>or</p> <p>ATBSDEPE</p> <p>or</p> <p>ATBSDFMU</p>	<p>Protect TP profile data sets and side information data sets.</p> <p>See <a href="#">“Giving Program Access to the APPC/MVS Administration Utility”</a> on page 149</p>
<p>In class VTAMAPPL, profiles with the following names:</p> <p>acbname</p>	<p>Protect ACBs.</p> <p>See <a href="#">“Controlling the Use of VTAM ACBs”</a> on page 143.</p>

---

## Part 5. System management



---

## Chapter 11. Operating APPC/MVS

MVS system commands, the APPC/MVS administration utility and dialog, and VTAM commands allow you to display information about, alter, or cancel the APPC/MVS environment and workload, as described in this section.

Only a few VTAM commands are briefly described here. Because VTAM commands cause varying degrees of network disruption, make sure you understand the implications of issuing them. See [z/OS Communications Server: SNA Operation](#) and, if necessary, [z/OS Communications Server: SNA Network Implementation Guide](#) for more information about all VTAM commands and their effects on a network.

References:

[z/OS MVS System Commands](#)

[z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#)

[z/OS MVS System Management Facilities \(SMF\)](#)

[z/OS Communications Server: SNA Network Implementation Guide](#)

[z/OS Communications Server: SNA Operation](#)

---

### Starting the APPC and ASCH Address Spaces

IBM supplies procedures in SYS1.PROCLIB that support starting the APPC and APPC/MVS transaction scheduler (ASCH) address spaces with the START command. You can modify these procedures to meet your installation's data processing requirements; for example, you may use names other than APPC and ASCH for the started procedures or for the jobnames of the APPC and ASCH address spaces. If you do so, however, remember that you must use these different procedure names or jobnames, instead of "APPC" or "ASCH", on subsequent START and CANCEL commands. Renaming the address spaces might cause problems if APPC/MVS tries to automatically restart one of its address spaces; see [“Restarting APPC/MVS”](#) on page 186 for more information about restart attempts.

The EXEC statements in the APPC and ASCH procedures shipped in SYS1.PROCLIB specify REGION=0K. If the installation does not modify the specified REGION=0K in an installation exit such as IEALIMIT or IEFUSI, the system gives the job step all the requested storage available below and above 16 megabytes. The resulting size of the region below and above 16 megabytes is unpredictable.

The IBM-supplied procedures specify REGION=0K because the APPC workload and configurations that each installation plans to run determine the amount of private storage that APPC requires. See [“Considering APPC/MVS Storage Requirements”](#) on page 210 for information about APPC/MVS storage usage.

To limit the region size of APPC/MVS, modify the REGION=0K parameter on the EXEC statement in the APPC and ASCH procedures.

To start the APPC and ASCH address spaces for APPC/MVS, issue the START (S) command to initialize each address space as follows:

```
START APPC,SUB=MSTR,APPC=xx
START ASCH,SUB=MSTR,ASCH=xx
```

**Note:** APPC or ASCH does not support the use of REUSASID=YES.

If you renamed the APPC or ASCH started procedures, specify the actual procedure name, instead of "APPC" or "ASCH" immediately after "START".

The xx represents the identifier for the parmlib member that contains initializing parameters. When no identifier is specified, the default for APPC is APPCPM00, and the default for ASCH is ASCHPM00. IBM does not supply default members with these names; however, sample APPCPMXX and ASCHPMXX members are in SYS1.SAMPLIB.

To specify more than one parmlib member after the START command, issue the commands as follows:

```
START APPC,SUB=MSTR,APPC=(xx,yy,...)
START ASCH,SUB=MSTR,ASCH=(xx,yy,...)
```

To start the APPC and ASCH address spaces automatically at IPL, an installation can add these START commands to the COMMNDxx member of the parmlib concatenation. The parmlib members used in the commands must be coded before the IPL.

## Restarting APPC/MVS

If APPC/MVS encounters an unrecoverable error when starting either of its two address spaces, APPC or ASCH, it attempts to restart the address space automatically. If your installation is using different names for the APPC or ASCH started procedure, APPC/MVS cannot use that name in its restart attempt; it uses "APPC" or "ASCH", which means that:

- The restart attempt fails if you have deleted or renamed the original APPC or ASCH procedure (instead of using a copy for your modifications), or
- The restart attempt successfully starts the address space, but through a procedure that you did not want to use. In this case, you may cancel the address space, and re-enter the START command, specifying the actual procedure name.

If you code a time on the PSTIMER parameter of the LUADD statement, VTAM keeps LU-to-LU sessions active during interruptions in APPC/MVS service. Even though APPC/MVS has been deactivated, the VTAM persistent sessions remain active for the length of time that you specify on the PSTIMER parameter. For an APPC/MVS LU that handles protected conversations, however, sessions on which syncpoint operations were in progress do not persist. All other sessions still have the persistent attribute. In this case, the session is unbound so that outstanding resynchronization work can proceed when the LU is reactivated.

If APPC/MVS could not restart the APPC address space automatically (indicated by message ATB006I), or if the address space was stopped by a CANCEL or FORCE command (message ATB010I or ATB012I, respectively), do the following to restart the APPC address space:

- Wait for the system to issue message ATB002I indicating that APPC has completed end processing.
- Enter a START APPC command.

If APPC/MVS could not restart the APPC/MVS transaction scheduler (ASCH) address space automatically (indicated by message ASB051I), or if the address space was stopped by a CANCEL or FORCE command (message ASB059I), do the following to restart the ASCH address space:

- Wait for the system to issue message ATB053I indicating that ASCH has completed end processing.
- Enter a START ASCH command.

Note that each time you cancel APPC and ASCH, the system marks the address spaces in which they were running as non-reusable until the next IPL. Therefore, do not cancel and then restart them unnecessarily.

See [“Recovering from APPC problems” on page 205](#) for more information on recovery from APPC problems.

## Displaying Information about APPC/MVS Work

An operator can display detailed information about APPC/MVS work running in the system by using the DISPLAY APPC (D APPC) and DISPLAY ASCH (D ASCH) commands. The information requested appears as ATB1xxI or ASB1xxI messages, respectively, which are described in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*. For the syntax of the DISPLAY command, see *z/OS MVS System Commands*.

The DISPLAY APPC command displays information about the status of TPs, APPC/MVS servers, and LUs. The DISPLAY ASCH command displays information about the status of the APPC/MVS transaction scheduler and the work it manages. Tables in [“Tracking Changes to the APPC/MVS Configuration and Workload” on page 196](#) summarize the types of information you can display with various DISPLAY commands.



## Dynamically Changing the APPC/MVS Environment

Through the SET command, your installation can add to or modify parmlib specifications for the APPC and ASCH address spaces, which affect side information files, and logical unit and scheduling characteristics. Also, through VTAM commands, you can alter the characteristics of APPC/MVS LUs.

### Changing Parmlib Specifications through the SET Command

You can use the SET command to:

- Change APPC address space parameters

```
SET APPC=xx
```

- Change APPC/MVS transaction scheduler parameters

```
SET ASCH=xx
```

When you want to include more than one parmlib member's specifications, separate each parmlib indicator with a comma and enclose them in parentheses.

```
SET APPC=(xx,yy,...) ASCH=(xx,yy,...)
```

When you use the SET command to change specifications for the APPC and ASCH address spaces, the parmlib member specified in the SET command does not cancel a previous parmlib member, but modifies it. When more than one parmlib member has been specified, the parmlib statements have a cumulative effect; that is, any one parmlib member might not reflect the current configuration.

For example, two parmlib members APPCPM1A and APPCPM2A contain parameters to add LUs.

```
LUADD
ACBNAME(LU1)
SCHED(ASCH)
BASE
TPDATA(SYS1.APPCTP)
TPLEVEL(SYSTEM)
```

*Figure 99. APPCPM1A*

```
LUADD
ACBNAME(LU2)
SCHED(ASCH)
TPDATA(SYS1.TESTTP)
TPLEVEL(USER)
```

*Figure 100. APPCPM2A*

If the parmlib member APPCPM1A is initialized on the START command when APPC is started, it is still in effect when APPCPM2A is initialized later by a SET command. Neither parmlib member reflects the current APPC configuration, because the configuration is a combination of the two parmlib members.

For more information about modifying the system environment with parmlib members, see:

- [“Stopping One or More LUs with the SET or VARY Command” on page 190,](#)
- [“Examples Using APPCPMxx Parmlib Members” on page 127,](#)
- [“Examples Using ASCHPMxx Parmlib members” on page 51, and](#)
- [“Tracking Changes to the APPC/MVS Configuration and Workload” on page 196.](#)

## Changing LU Characteristics through VTAM Commands

You can use VTAM commands to change the status of network resources. The MODIFY CNOS command, for example, changes the session limits between an APPC/MVS LU and its partner LU. If you decrease the session limits between LUs, you might impact the performance of transaction programs that use those LUs.

See [z/OS Communications Server: SNA Operation](#) for more information about MODIFY CNOS and other VTAM commands.

## Stopping APPC/MVS Work

You can stop APPC/MVS work directly or indirectly in a variety of ways, which include:

- Deactivating a TP through its TP profile
- Stopping an initiator address space with the STOP command
- Stopping a class of transaction programs with the SET command
- Stopping an LU with the SET or VARY command
- Stopping a TP or APPC/MVS address space with the CANCEL command
- Stopping VTAM with the HALT command

Some methods gradually stop (quiesce) the work, while others immediately stop the work. Depending on the circumstances, you might want to use several methods in sequence, to ensure work is stopped as cleanly or quickly as possible.

Table 24 on page 194 summarizes the methods of stopping APPC/MVS work.

## Deactivating a Transaction Program through its TP Profile

You can control the running of a transaction program from its TP profile. To prevent a transaction program from being scheduled or to stop new requests for a TP, run the APPC administration utility with the TP MODIFY command to change the active status of the TP to NO. You can also modify a TP profile by using the APPC administration dialog.

Active status is controlled by the ACTIVE keyword in the TP profile; its value can be either YES or NO. The following example shows the TP MODIFY utility command that deactivates a TP named MAIL.

```
//STEP      EXEC  PGM=ATBDSDFMU
//SYSPRINT DD   SYSOUT=A
//SYSSDLIB DD   DSN=SYS1.APPCTP,DISP=SHR
//SYSSDOUT DD   SYSOUT=*
//SYSIN     DD   DATA
TPMODIFY
  TPNAME(MAIL)
  SYSTEM
  ACTIVE(NO)
/*
```

Figure 101. Example of Deactivating a TP

If the TP is running when you change its active status from YES to NO, the TP instance that is running is allowed to complete, and all queued requests are also allowed to complete. No new requests for the TP, however, are allowed.

For information about the APPC administration utility, see [Chapter 6, “Using the APPC/MVS Administration Utility,”](#) on page 73.

# Stopping an Initiator Address Space with the STOP Command

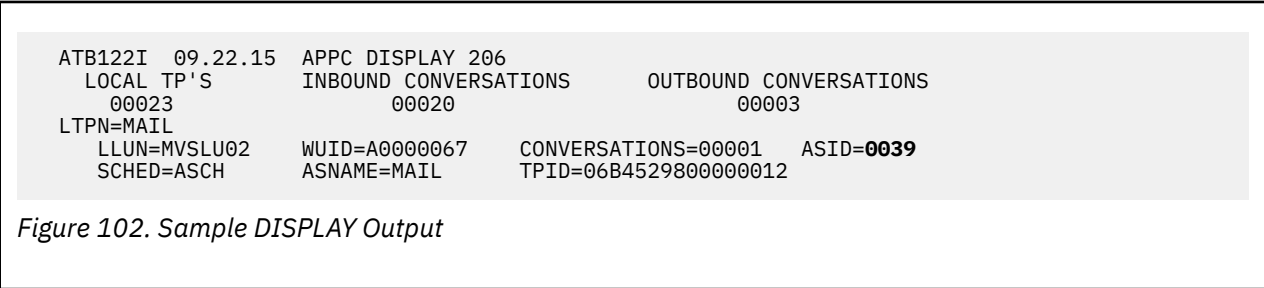
The STOP command stops an APPC/MVS transaction initiator through its address space name (ASCHINT) and address space identifier. All initiators for the APPC/MVS transaction scheduler use the name ASCHINT.

```
STOP ASCHINT,A=asid
```

To find out the address space identifier of the TP's transaction initiator, issue the DISPLAY command as follows. This example assumes the local TP name is MAIL.

```
DISPLAY APPC,TP,LIST,LTPN=MAIL
```

This command results in output that might look like the following:



To make sure the TP is actually running in a transaction initiator and is not queued, check the ASNAME keyword. If the value is ASCH, the TP is queued. If the TP is not queued, note the address space identifier value following the ASID keyword. Use this value in the STOP command, as follows:

```
STOP ASCHINT,A=0039
```

## Note:

1. If there is a problem within the TP, you might need to cancel the TP by issuing a CANCEL command. For information about the CANCEL command, see [“Stopping a TP or APPC/MVS Address Space with the CANCEL Command”](#) on page 192.
2. Another initiator might be created for the next TP request depending on the MIN/MAX ratio.
3. To end both the initiator and the TP, issue a STOP command and a CANCEL command. For information about the CANCEL command, see [“Stopping a TP or APPC/MVS Address Space with the CANCEL Command”](#) on page 192.

# Stopping a Class of Transaction Programs with the SET Command

A way to stop a class of transaction programs is to issue a SET command specifying an ASCHPMxx parmlib member that deletes the class definition.

When you delete a class, APPC/MVS:

- Rejects new inbound and outbound Allocate requests for TPs in the class
- Either quiesces or immediately stops existing work:
  - When the CLASSDEL statement does not contain the WORKQ keyword, or contains WORKQ=DRAIN, the work is quiesced. That is, the TP currently running and all TPs on the work queue for the class are allowed to complete their processing before APPC/MVS deletes the class.
  - When CLASSDEL contains WORKQ=PURGE, all TPs on the work queue are rejected; only the currently running TP is allowed to complete its processing.

For example, to delete a class named TEST and drain its work queue, create a parmlib member ASCHPM4D with the following information:

```
CLASSDEL  
CLASSNAME (TEST)
```

*Figure 103. ASCHPM4D*

To activate the parmlib member and delete the class, issue the following SET command:

```
SET ASCH=4D
```

## Stopping One or More LUs with the SET or VARY Command

You can stop an LU, and the work associated with it, by issuing the SET command to specify an APPCPMxx parmlib member that contains an LUDEL statement, or by issuing VTAM's VARY command. You might want to stop an LU in the following situations:

- The LU is not functioning properly (for example, because of a VTAM error).
- You wish to move a particular LU to another system in the sysplex for workload balancing, to perform maintenance on a system without a disruption in service, or for other availability reasons.
- Your installation has designated a certain TP\_Profile file to be used by one or more LUs, and you want to stop the TPs defined in that data set.
- You want to stop a particular scheduler.

Issuing the SET command quiesces the APPC work for the LU; issuing the VARY command immediately stops the work.

### Using the SET Command to Quiesce Work

To use this method of stopping APPC work, issue the SET command to specify an APPCPMxx parmlib member that contains one LUDEL statement for each LU to be deleted. For example, if you wanted to delete an LU named *MYLU*, you would first code a parmlib member such as *APPCPM1D* in the following example.

```
LUDEL  
ACBNAME (MYLU)
```

*Figure 104. APPCPM1D*

After coding the parmlib member, issue the following SET command.

```
SET APPC=1D
```

After you activate the parmlib member through the SET command, APPC/MVS:

- Rejects any new conversations (that is, inbound or outbound Allocate requests) to or from the LU specified in the LUDEL statement.
- Allows each existing conversation to continue without interruption. Existing conversations are those for which — at the very least — one LU has successfully sent and its partner LU has successfully received the Allocate request.
- Issues message ATB051I to hardcopy, to indicate that the LU has been deleted from the configuration.
- Deletes the LU's association with a VTAM generic resource name, if any.
- Closes the ACB for the LU, as soon as existing conversations have ended. If the LU handles protected conversations, APPC/MVS defers resynchronization processing for incomplete units of recovery, if any.

By adding the PERSIST keyword on the LUDEL statement for a LU, APPC/MVS will not deactivate any persistent sessions between the LU and its partners. This will allow you to add this LU back into the

configuration later, yet not lose any of the sessions that were active at the time of the LUDEL. This assumes that the LU is added back before the persistent sessions time limit expires that was specified on the PSTIMER keyword on the LUADD statement.

At this point, you may restart the LU by issuing a SET command that specifies an LUADD statement for this LU.

**Note:**

If the PERSIST keyword was specified on the previous LUDEL, you must start the LU with the same NQN capability as the original LUADD for this LU. Failure to do so could cause the LU to not become active and an ATB052E error message with a Reason Code of X'78'.

You may choose to re-add this LU to another image in your sysplex, as long as you have the equivalent VTAM definitions and connectivity, and access to an equivalent TP Profile data set and scheduler.

If you have configured your sysplex to support Multi-Node Persistent Sessions (MNPS), you can add this LU on a different image in the sysplex AND still keep all previous sessions the LU had prior to issuing the LUDEL with the PERSIST keyword.

Because LUDEL processing quiesces work, rather than stopping it immediately, APPC/MVS might not close the ACB for the LU as quickly as you want. To avoid delays, you may eliminate queued work or stop running work **before** issuing the SET command to process an LUDEL statement. For example, you may stop a class of TPs as described in [“Stopping a Class of Transaction Programs with the SET Command”](#) on page 189.

If LUDEL processing has already started, but is proceeding too slowly, you can issue VTAM's VARY TERM command to accelerate the process by terminating one or more sessions for the LU.

## Using the VARY Command to Stop Work Immediately

VTAM's VARY TERM and VARY INACT commands allow you to shut down an LU and its work almost immediately. For APPC/MVS LUs, IBM recommends that you use these commands in sequence:

1. Issue VARY TERM to terminate all sessions for a specified APPC/MVS LU. When you issue VARY TERM, all TPs involved in existing conversations that are processed through VTAM receive control.

This interruption prevents TPs from hanging, if they are waiting for an outstanding APPC/MVS or CPI-C call (for example, a TP might be waiting for its partner TP's response to a Confirm call).

2. Issue VARY INACT without the TYPE parameter to deactivate the specified APPC/MVS LU. When you issue VARY INACT:
  - Either VTAM or APPC/MVS rejects new conversations for the LU
  - APPC/MVS issues message ATB051I to hardcopy, to indicate that the LU has been deleted from the configuration.
  - APPC/MVS immediately closes the ACB for the LU. If the LU handles protected conversations:
    - Units of recovery for those conversations are put into backout-required state;
    - APPC/MVS defers resynchronization processing for incomplete units of recovery, if any; and
    - APPC/MVS unregisters the LU with RRS.

At this point, you may reactivate the LU, so it can begin to process new conversations. Once the LU re-registers with RRS, the LU can resume resynchronization processing for incomplete units of recovery, if any.

If you issue VARY INACT without VARY TERM, or issue VARY INACT with the TYPE operand, work is stopped quickly, but not as cleanly. If any TPs are waiting for an outstanding APPC/MVS or CPI-C call, they might hang indefinitely. For VARY INACT with TYPE=IMMED or TYPE=UNCOND, APPC/MVS cannot return control to such TPs, so you risk more than hanging some work; data loss and resource contention are only some of the possible consequences. Use VARY TERM and VARY INACT in sequence, to make sure you stop work quickly but cleanly.

For VARY command syntax, see [z/OS Communications Server: SNA Operation](#).

# Stopping a TP or APPC/MVS Address Space with the CANCEL Command

The CANCEL (C) command immediately stops either:

- A single instance of a TP
- The APPC or ASCH address space.

If you cannot cancel an APPC TP and it is imperative that you stop it, you can issue the FORCE command. Before issuing FORCE, refer to [z/OS MVS System Commands](#) for a list of restrictions concerning this command.

## Stopping a Single Instance of a TP

To immediately stop a particular TP running in a particular address space, you must know the jobname for the TP and its address space identifier (ASID). If you know the TP name (for example, MAIL) and the user ID that invoked it (for example, JOHN), you can issue one of the following DISPLAY commands to find out the jobname and the ASID.

```
DISPLAY ASCH,ALL,LTPN=MAIL
DISPLAY ASCH,ALL,USERID=JOHN
```

These DISPLAY commands produce output like the following:

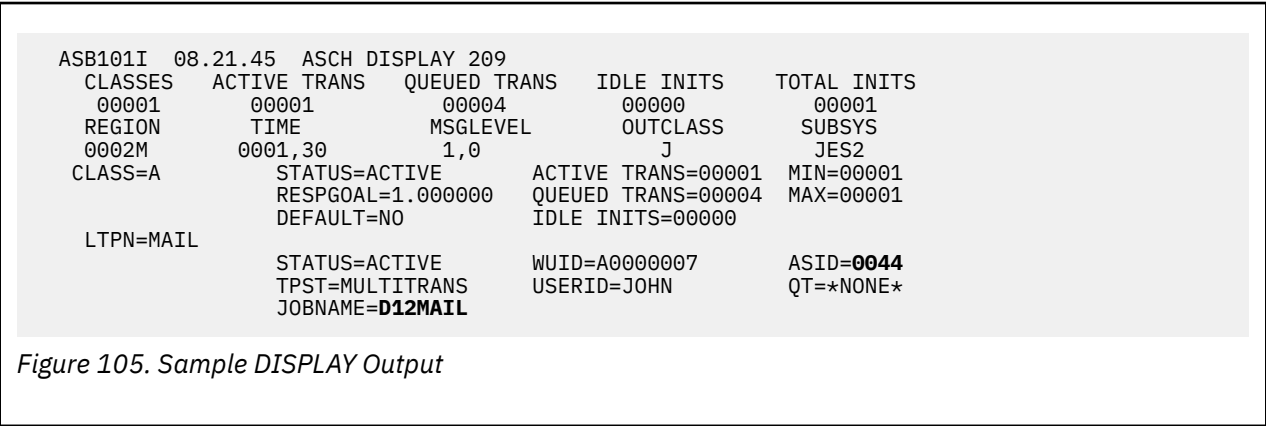


Figure 105. Sample DISPLAY Output

To immediately stop the TP that is processing John's mail, use these values in the CANCEL command:

```
CANCEL D12MAIL,A=0044
```

## Stopping the APPC and ASCH Address Spaces

Because the APPC and ASCH address spaces are started by cataloged procedures specified on a START command, you can immediately stop them with the CANCEL command, as follows:

```
CANCEL APPC
CANCEL ASCH
```

If your installation is using different jobnames for the APPC and ASCH address spaces, specify the actual jobname on the CANCEL command.

### Note:

1. Cancelling the ASCH address space immediately ends all running or queued TPs scheduled through the APPC/MVS transaction scheduler. Outbound Allocate requests from address spaces not associated with ASCH can be processed only if a NOSCHED system base LU exists.
2. Cancelling the APPC address space immediately ends all APPC/MVS TPs, schedulers, servers, and LUs, which can have drastic results.

3. Each time you cancel APPC and ASCH, the system marks the address spaces in which they were running as non-reusable until the next IPL. Therefore, do not cancel and then restart them unnecessarily.

## Stopping VTAM with the HALT Command

Although other methods of stopping APPC/MVS work also might affect partner systems, VTAM's HALT command affects the entire network, which is much more disruptive. Do not issue a HALT command when your sole objective is to stop APPC/MVS work. *z/OS Communications Server: SNA Network Implementation Guide* describes the circumstances under which you may issue a HALT command; read the sections on halting and cancelling VTAM to make sure you thoroughly understand the implications of using HALT.

Unlike other methods of stopping work, a HALT command affects **all** of the APPC/MVS LUs that are defined to VTAM, not just selected LUs. [Table 23 on page 193](#) describes how each type of HALT command affects APPC/MVS work.

Table 23. VTAM's HALT Command and Its Effect on APPC/MVS Work	
When you issue:	This is what happens to APPC/MVS work:
HALT	<ul style="list-style-type: none"> <li>• APPC/MVS closes the LU's ACB immediately.</li> <li>• New conversations are rejected.</li> <li>• All TPs involved in existing conversations that are processed through VTAM receive control; this interruption prevents TPs from hanging, if they are waiting for an outstanding APPC/MVS or CPI-C call.</li> <li>• If the LU handles protected conversations: <ul style="list-style-type: none"> <li>– Units of recovery for those conversations are put into backout-required state;</li> <li>– APPC/MVS defers resynchronization processing for incomplete units of recovery, if any; and</li> <li>– APPC/MVS unregisters the LU with RRS.</li> </ul> </li> </ul> <p>Issuing HALT allows you to cleanly end existing work, and to restart the LU as soon as you have restarted VTAM. Once the LU re-registers with RRS, it can resume resynchronization processing for incomplete units of recovery, if any.</p>
HALT QUICK	<ul style="list-style-type: none"> <li>• APPC/MVS closes the LU's ACB immediately.</li> <li>• New conversations are rejected.</li> <li>• Existing conversations that are processed through VTAM fail when a TP issues the next APPC/MVS or CPI-C service call that requires APPC/MVS and VTAM interaction. If any TPs are waiting for an outstanding APPC/MVS or CPI-C call to complete processing, they <b>might</b> not have control returned to them. Such TPs might hang indefinitely.</li> <li>• If the LU handles protected conversations: <ul style="list-style-type: none"> <li>– Units of recovery for those conversations are put into backout-required state;</li> <li>– APPC/MVS defers resynchronization processing for incomplete units of recovery, if any; and</li> <li>– APPC/MVS unregisters the LU with RRS.</li> </ul> </li> </ul> <p>Issuing HALT QUICK is similar to issuing VARY INACT with the TYPE parameter: it accelerates the process of closing the LU, but might not allow conversations to end cleanly. Thus, you benefit from the quickness of close processing — you can restart LUs immediately after you bring up VTAM again. The possible risk is indefinitely hanging existing APPC/MVS work.</p>

Table 23. VTAM's HALT Command and Its Effect on APPC/MVS Work (continued)

When you issue:	This is what happens to APPC/MVS work:
HALT CANCEL	<ul style="list-style-type: none"> <li>• APPC/MVS closes the LU's ACB immediately.</li> <li>• New conversations are rejected.</li> <li>• Existing conversations that are processed through VTAM fail when a TP issues the next APPC/MVS or CPI-C service call that requires APPC/MVS and VTAM interaction. If any TPs are waiting for an outstanding APPC/MVS or CPI-C call, they <b>cannot</b> have control returned to them.</li> <li>• If the LU handles protected conversations: <ul style="list-style-type: none"> <li>– Units of recovery for those conversations are put into backout-required state;</li> <li>– APPC/MVS defers resynchronization processing for incomplete units of recovery, if any; and</li> <li>– APPC/MVS unregisters the LU with RRS.</li> </ul> </li> </ul> <p>Issuing HALT CANCEL is similar to HALT QUICK in that you may restart APPC/MVS LUs immediately after bringing up VTAM again. However, for HALT CANCEL, APPC/MVS is unable to end its work cleanly, so you risk more than hanging some existing work. Because TPs do not receive control again, data loss and resource contention are some of the possible consequences of stopping VTAM this abruptly.</p>

## Summary of Methods of Stopping APPC/MVS Work

Table 24. Stopping APPC/MVS Work

When you want to:	Use one of the following:	Which affects:	By:
Prevent a transaction program (TP) from running	APPC/MVS utility or dialog to deactivate a TP through its profile	All new conversations for one specific TP	Rejecting inbound Allocate requests for the TP
Stop a currently running TP	CANCEL command	The currently running instance of a TP, and its partner TPs	Abnormally ending the TP
	STOP command	Same as for CANCEL	Abnormally ending the initiator address space in which the TP is running
Stop a class of TPs	SET ASCH command, with ASCHPMxx that contains CLASSDEL with WORKQ=DRAIN	All TPs defined to a specific class	Quiescing running or queued TPs, and deleting the class after the running and queued TPs have ended
	SET ASCH command, with ASCHPMxx that contains CLASSDEL with WORKQ=PURGE	Same as for CLASSDEL with WORKQ=DRAIN	Rejecting all queued TPs, allowing only the currently running TP to finish its processing, and deleting the class after the one TP has ended



Table 24. Stopping APPC/MVS Work (continued)

When you want to:	Use one of the following:	Which affects:	By:
Stop a logical unit (LU)	SET APPC command, with APPCPMxx that contains LUDEL	All inbound or outbound conversations for the LU	Rejecting new conversations, but allowing active or queued (that is, existing) conversations to continue processing without interruption
	VARY TERM command, followed by VARY INACT command	All inbound or outbound conversations for the LU	Rejecting new conversations and interrupting all TPs involved in existing conversations that are processed through VTAM
	VARY INACT command, with the TYPE operand	Same as for VARY TERM with VARY INACT	Rejecting new conversations, and failing existing conversations on the next APPC/MVS or CPI-C service that requires APPC/MVS and VTAM interaction. Not recommended, because TPs waiting for an outstanding call to complete might hang indefinitely. IBM recommends that you issue the VARY TERM command before issuing VARY INACT to avoid hanging TPs.
Stop the ASCH address space	CANCEL ASCH command	All conversations for TPs scheduled through the APPC/MVS scheduler	Immediately ending all running or queued TPs scheduled through the APPC/MVS transaction scheduler. Outbound Allocate requests from address spaces not associated with ASCH can be processed only if a NOSCHED system base LU exists.
Stop the APPC address space	CANCEL APPC command	All APPC/MVS work on this MVS system	Immediately ending all APPC/MVS TPs, schedulers, servers, and LUs
Stop VTAM	HALT command	All TPs, regardless of the operating system on which they reside, that require VTAM services to communicate	Rejecting new conversations and interrupting all TPs involved in existing conversations that are processed through VTAM
	HALT QUICK command	Same as for HALT	Rejecting new conversations and failing existing conversations that are processed through VTAM on the next APPC/MVS or CPI-C service that requires APPC/MVS and VTAM interaction. If any TPs are waiting for an outstanding APPC/MVS or CPI-C call, they might hang.
	HALT CANCEL command	Same as for HALT	Same as for HALT QUICK, except that TPs waiting for outstanding calls do not receive control again.

## Tracking Changes to the APPC/MVS Configuration and Workload

During APPC/MVS initialization and processing, the system issues ASBxxx and ATBxxx messages to the master console, or a console with master authority. These messages primarily report only errors, or status changes that might require operator intervention. Messages that report successful processing generally are routed to hardcopy, thus reducing message traffic on the console. *z/OS MVS System Messages, Vol 3 (ASB-BPX)* describes the ASBxxx and ATBxxx messages.

This message design, along with the cumulative effects of parmlib members and possible command processing delays, might require you to rely on the DISPLAY command to obtain current, detailed information about the APPC/MVS configuration and workload. To tailor information to your specific needs, enter the DISPLAY APPC and DISPLAY ASCH commands, as shown in summary tables beginning with “Displaying TP Status” on page 196, with operands that filter the information returned to the console. The underlined keywords are defaults; you can omit them when you issue the command.

To monitor SET command changes to APPC/MVS parmlib members, you can also review the contents of SMF type 90 records. Every time a SET command is issued, SMF records the parmlib information associated with the SET. The SMF type 90 record is useful for tracking changes to the APPC configuration. *z/OS MVS System Management Facilities (SMF)* describes the format and contents of type 90 records.

### Displaying TP Status

Table 25. Displaying TP Status	
To Display	Issue Command
Number of local TPs and number of inbound and outbound conversations	DISPLAY APPC, <u>TP</u> ,SUMMARY
Number of local TPs, number of inbound and outbound conversations, and specific information about each local TP	DISPLAY APPC, <u>TP</u> ,LIST
Number of local TPs, number of inbound and outbound conversations, specific information about each local TP, and specific information about each partner TP and its local conversation.	DISPLAY APPC, <u>TP</u> ,ALL
Local TP in a particular address space	DISPLAY APPC, <u>TP</u> ,[LIST ALL], ASID=asid
Inbound or outbound conversations	DISPLAY APPC, <u>TP</u> ,[LIST ALL], DIR=[IN OUT]
Conversations with idle time equal to or greater than a time specified in seconds	DISPLAY APPC, <u>TP</u> ,[LIST ALL], IT=sssss
Conversations using a particular local LU	DISPLAY APPC, <u>TP</u> ,[LIST ALL], LLUN=lluname
Information about a particular local TP	DISPLAY APPC, <u>TP</u> ,[LIST ALL], LTPN=ltpname
TPs running because of an allocate request from a particular user ID.	DISPLAY APPC, <u>TP</u> ,[LIST ALL], USERID=userid
Conversations using a particular partner LU that resides on a particular network	DISPLAY APPC, <u>TP</u> ,[LIST ALL], PNET=pnetid, PLUN=pluname
Information about a particular partner TP	DISPLAY APPC, <u>TP</u> ,[LIST ALL], PTPN=ptpname
TPs scheduled by a particular transaction scheduler	DISPLAY APPC, <u>TP</u> ,[LIST ALL], SCHED=schedname
TPs not associated with a particular transaction scheduler	DISPLAY APPC, <u>TP</u> ,[LIST ALL], SCHED=*NONE*
Information about a particular served TP	DISPLAY APPC, <u>TP</u> ,[LIST ALL], STPN=stpname
Information about a particular non-served TP	DISPLAY APPC, <u>TP</u> ,[LIST ALL], LTPN=ltpname

## Displaying UR Status

Table 26. Displaying UR Status	
To Display	Issue Command
Number of URs related to protected conversations between APPC/MVS TPs and their partner TPs	DISPLAY APPC,UR, <u>SUMMARY</u>
Number of URs related to protected conversations between APPC/MVS TPs and their partner TPs, followed by one list of information for each unit of recovery, represented by a UR identifier (URID)	DISPLAY APPC,UR,LIST
Number of URs related to protected conversations between APPC/MVS TPs and their partner TPs, followed by one list of information for each unit of recovery, which includes another list of information about each APPC/MVS expression of recoverable interest for that UR	DISPLAY APPC,UR,ALL
Information about URs represented by a URID.	DISPLAY APPC,UR,[LIST ALL], URID= <i>urid</i>
Information about a particular UR, represented by a logical unit of work identifier (LUWID)	DISPLAY APPC,UR,[LIST ALL], LUWID= <i>luwid</i>
Number of URs related to protected conversations between APPC/MVS TPs and their partner TPs, for a particular partner LU that resides on a particular network	DISPLAY APPC,UR,[LIST ALL], PNET= <i>pnetid</i> , PLUN= <i>pluname</i>
Number of URs related to protected conversations between APPC/MVS TPs and their partner TPs, for a particular local LU	DISPLAY APPC,UR,[LIST ALL], LLUN= <i>lluname</i>

## Displaying Server Status

Table 27. Displaying Server Status	
To Display	Issue Command
Number of servers, number of allocate queues, and total number of queued allocate requests in the system.	DISPLAY APPC,SERVER, <u>SUMMARY</u>
Number of servers, number of allocate queues, total number of queued allocate requests in the system, and specific information about each allocate queue.	DISPLAY APPC,SERVER,LIST
Number of servers, number of allocate queues, total number of queued allocate requests in the system, specific information about each allocate queue, and specific information about server.	DISPLAY APPC,SERVER,ALL
Information about a server with a given address space identifier, or ASID.	DISPLAY APPC,SERVER,[LIST ALL], ASID= <i>asid</i>
Information about a server with a given address space name, or ASNAME.	DISPLAY APPC,SERVER,[LIST ALL], ASNAME= <i>asname</i>
Allocate queues serving and servers registered for a particular local LU.	DISPLAY APPC,SERVER,[LIST ALL], LLUN= <i>lluname</i>

Table 27. Displaying Server Status (continued)	
To Display	Issue Command
Allocate queues serving and servers registered for a particular TP name.	DISPLAY APPC,SERVER,[LIST ALL], STPN= <i>stpname</i>

## Displaying LU Status

Table 28. Displaying LU Status	
To Display	Issue Command
Number of active, outbound, pending, and terminating LUs, and the side information file name	DISPLAY APPC,LU, <u>SUMMARY</u>
Number of active, outbound, pending, and terminating LUs, the side information file name, and specific information about local LUs	DISPLAY APPC,LU,LIST
Number of active, outbound, pending, and terminating LUs; the side information file name; specific information about local LUs; and partner LU names	DISPLAY APPC,LU,ALL
Information about a particular local LU	DISPLAY APPC,LU,[LIST ALL], LLUN= <i>lluname</i>
Information about local LUs for which session limits have been established with a particular partner LU that resides on a particular network	DISPLAY APPC,LU,[LIST ALL], PNET= <i>pnetid</i> , PLUN= <i>pluname</i>
LUs controlled by a particular transaction scheduler	DISPLAY APPC,LU,[LIST ALL], SCHED= <i>schedname</i>
LUs that are not associated with a particular transaction scheduler.	DISPLAY APPC,LU,[LIST ALL], SCHED= <i>*NONE*</i>

## Displaying Scheduling Status

Table 29. Displaying Scheduling Status	
To Display	Issue Command
Number of classes, active and queued transactions, idle and total initiators; and default scheduling information specified in an ASCHPMxx parmlib member	DISPLAY ASCH, <u>SUMMARY</u>
In addition to the information displayed when you use the SUMMARY keyword, this displays specific information about each class	DISPLAY ASCH,LIST
In addition to the information displayed when you use the LIST keyword, this displays specific information about each active and queued TP in each class	DISPLAY ASCH,ALL
Scheduling information about a local TP in a particular address space	DISPLAY ASCH,[LIST ALL], ASID= <i>asid</i>
Scheduling information about a particular class	DISPLAY ASCH,[LIST ALL], CLASS= <i>classname</i>
Scheduling information about local TPs with a queue time equal to or greater than a time specified in seconds	DISPLAY ASCH,[LIST ALL], QT= <i>sssss</i>

Table 29. Displaying Scheduling Status (continued)	
To Display	Issue Command
Scheduling information about local TPs running due to an allocate request from a particular user ID	DISPLAY ASCH,[LIST ALL], USERID= <i>userid</i>
Scheduling information about local TPs with a schedule type of standard or multi-trans	DISPLAY ASCH,[LIST ALL], TPST=[STANDARD MULTITRANS]

## Examples Using the DISPLAY Command

The following examples illustrate the results of DISPLAY APPC commands with various keywords and filters.

### DISPLAY APPC,LU

To show the current configuration of APPC/MVS LUs, enter the following command on one system:

```
D APPC,LU,ALL
```

This command produces output like the following:

```

ATB121I 15.55.45 APPC DISPLAY      FRAME 1  F  E  SYS=SY1
ACTIVE LU'S   OUTBOUND LU'S   PENDING LU'S   TERMINATING LU'S
00008        00000        00000        00000
SIDEINFO=SYS1.APPCSI
LLUN=Z098AP01  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU1  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
LLUN=Z098AP02  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00000  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU4
LLUN=Z096AP02  SCHED=*NONE*     BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU4
LLUN=Z096AP03  SCHED=ASCH      BASE=YES       NQN=YES
STATUS=ACTIVE  PARTNERS=00003  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU1  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU
PLUN=USIBMY0.MVSLU
PLUN=USIBMY0.MVSLU4
LLUN=Z098AP04  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00000  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
LLUN=Z096AP04  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001  TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU1  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
LLUN=Z0A4AP03  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00001  TPLEVEL=SYSTEM SYNCPT=YES
GRNAME=*NONE*  RMNAME=ATB.USIBMY0.Z0A4AP03.IBM
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.Z0A4AP04
LLUN=Z0A4AP04  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00002  TPLEVEL=SYSTEM SYNCPT=YES
GRNAME=*NONE*  RMNAME=ATB.USIBMY0.Z0A4AP04.IBM
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.Z0B4AP01
PLUN=USIBMY0.Z0B4AP04

```

Figure 106. Sample DISPLAY Output for System SY1 on Network USIBMY0

According to the output:

- The last two LUs are capable of handling protected conversations, as illustrated by the display field SYNCPT=YES. These LUs are registered with RRS as resource managers, with the resource manager names shown in the RMNAME field. You can use these RMNAME values on the RRS ISPF panel interface when you want to:
  - Obtain the resource manager token and state.
  - Obtain a list of units of recovery (URs) for the resource manager. (An alternative method of obtaining UR information is using the DISPLAY APPC,UR command with LLUN as a filter keyword.)
  - Remove a resource manager's expression of interest in a UR.
- Two LUs, the first and fourth, are members of the same VTAM generic resource group named MVSLU1.
- The fourth LU, Z096AP03, has partners that are also members of different generic resource groups. As shown by their network-qualified names, some of those partners reside on another system, USIBMZ0.

To display information about the partners on the USIBMZ0 system, issue the same command on USIBMZ0:

```
D APPC,LU,ALL
```

This command produces output like the following:

```
ATB121I 15.48.39 APPC DISPLAY      FRAME 1  F   E  SYS=SY2
ACTIVE LU'S   OUTBOUND LU'S   PENDING LU'S   TERMINATING LU'S
00003        00000        00000        00000
SIDEINFO=SYS1.APPCSI
LLUN=Z098AP01  SCHED=ASCH      BASE=NO        NQN=YES
STATUS=ACTIVE  PARTNERS=00000    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=*NONE*  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
LLUN=Z098AP02  SCHED=ASCH      BASE=YES        NQN=NO
STATUS=ACTIVE  PARTNERS=00001    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
LLUN=Z098AP04  SCHED=ASCH      BASE=NO        NQN=NO
STATUS=ACTIVE  PARTNERS=00002    TPLEVEL=SYSTEM SYNCPT=NO
GRNAME=MVSLU4  RMNAME=*NONE*
TPDATA=SYS1.APPCTP
PLUN=USIBMY0.MVSLU1
PLUN=USIBMY0.Z096AP02
```

Figure 107. Sample DISPLAY Output for System SY2 on Network USIBMZ0

According to this output, LUs on the two systems share the same TP profile and side information data sets. Some restrictions apply to sharing these data sets among systems; see [“Restrictions on Invoking the APPC/MVS Administration Utility”](#) on page 80 in [Chapter 6, “Using the APPC/MVS Administration Utility,”](#) on page 73 for further details.

## DISPLAY APPC,TP

To investigate existing APPC/MVS work, you can issue the DISPLAY APPC command to see all TPs currently executing or awaiting execution.

```
DISPLAY APPC,TP,ALL
```

This command produces output like the following:

```

D APPC,TP,ALL
ATB122I 19.26.54 APPC DISPLAY 422
LOCAL TP'S      INBOUND CONVERSATIONS      OUTBOUND CONVERSATIONS
00003           00002           00002
LTPN=*UNKNOWN*
  LLUN=Z0A4AP03  WUID=*UNKNOWN*  CONVERSATIONS=00001  ASID=0016
  SCHED=*NONE*   ASNAME=DBUTLER  TPID=0622891000000002
PTPN=TBDRIVER
  PLUN=USIBMZ0.MVSLU4
  PROTECTED=YES  USERID=DBUTLER  DIRECTION=OUTBOUND
  VERBS=00000006 IT=040.662S
  MODE=TRANPAR  VTAMCID=01000005 SYNC POINT IN PROG=YES
  LUWID=USIBMY0.Z0A4AP03 4AAE3101D784 0001
LTPN=TBDRIVER
  LLUN=Z0A4AP04  WUID=A0000001  CONVERSATIONS=00002  ASID=0022
  SCHED=ASCH     ASNAME=VSTEST  TPID=06228A7000000004
PTPN=*UNKNOWN*
  PLUN=USIBMZ0.Z0A4AP03
  PROTECTED=YES  USERID=DBUTLER  DIRECTION=INBOUND
  VERBS=00000003 IT=*NONE*
  MODE=TRANPAR  VTAMCID=01000006 SYNC POINT IN PROG=NO
  LUWID=USIBMY0.Z0A4AP03 4AAE3101D784 0001
PTPN=TBDRIVER
  PLUN=USIBMZ0.MVSLU3
  PROTECTED=YES  USERID=*NONE*   DIRECTION=OUTBOUND
  VERBS=00000003 IT=*NONE*
  MODE=TRANPAR  VTAMCID=0100000B SYNC POINT IN PROG=NO
  LUWID=USIBMY0.Z0A4AP03 4AAE3101D784 0001
LTPN=TBDRIVER
  LLUN=Z0A4AP03  WUID=A0000002  CONVERSATIONS=00001  ASID=0041
  SCHED=ASCH     ASNAME=VSTEST  TPID=06228BD000000006
PTPN=*UNKNOWN*
  PLUN=USIBMZ0.MVSLU4
  PROTECTED=YES  USERID=*NONE*   DIRECTION=INBOUND
  VERBS=00000002 IT=*NONE*
  MODE=TRANPAR  VTAMCID=0100000C SYNC POINT IN PROG=NO
  LUWID=USIBMY0.Z0A4AP03 4AAE3101D784 0001

```

Figure 108. Sample DISPLAY Output

According to the output:

- All of the TPs have protected conversations with their partner TPs.
- For one outbound conversation, a syncpoint operation (either Commit or Backout) is in progress.

## DISPLAY APPC,UR

To determine whether any APPC/MVS protected conversations require or are undergoing resynchronization processing, enter the following command:

```
D APPC,UR,ALL
```

This command produces output like the following:

```

ATB104I 19.33.55 APPC DISPLAY 431
  APPC UR'S                      EXPRESSIONS OF INTEREST
    00003                        00004
URID=AC964AAE7F36A0000000000101010000
  EXPRESSION OF INTEREST COUNT=00001      SYNC POINT IN PROG=YES
  LUWID=USIBMZ0.Z0A4AP03 4AAE3101D784 0001
  LTPN=*UNKNOWN*
  PTPN=TBDRIVER
    CONV CORRELATOR=062313F800000001
    PLUN=USIBMZ0.MVSLU4      LLUN=Z0A4AP03      DIRECTION=OUTBOUND
    RESYNC REQUIRED=NO        IMPLIED FORGET=NO
URID=AC964ABD7F36A22800000000201010000
  EXPRESSION OF INTEREST COUNT=00002      SYNC POINT IN PROG=NO
  LUWID=USIBMZ0.Z0A4AP03 4AAE3101D784 0001
  LTPN=TBDRIVER
  PTPN=TBDRIVER
    CONV CORRELATOR=06231CC800000005
    PLUN=USIBMZ0.MVSLU3      LLUN=Z0A4AP04      DIRECTION=OUTBOUND
    RESYNC REQUIRED=NO        IMPLIED FORGET=NO
  LTPN=TBDRIVER
  PTPN=*UNKNOWN*
    CONV CORRELATOR=062313F800000001
    PLUN=USIBMZ0.Z0A4AP03      LLUN=Z0A4AP04      DIRECTION=INBOUND
    RESYNC REQUIRED=NO        IMPLIED FORGET=NO
URID=AC964AD07F36A45000000000301010000
  EXPRESSION OF INTEREST COUNT=00001      SYNC POINT IN PROG=NO
  LUWID=USIBMZ0.Z0A4AP03 4AAE3101D784 0001
  LTPN=TBDRIVER
  PTPN=*UNKNOWN*
    CONV CORRELATOR=06231CC800000005
    PLUN=USIBMZ0.MVSLU4      LLUN=Z0A4AP03      DIRECTION=INBOUND
    RESYNC REQUIRED=NO        IMPLIED FORGET=NO

```

Figure 109. Sample DISPLAY Output

According to the output:

- The first unit of recovery, which represents the outbound conversation that appears in “[DISPLAY APPC,TP](#)” on page 200, does not require resynchronization processing.
- The URID and logical work unit identifier (LUWID) values can serve as input for the RRS ISPF panel interface.

## DISPLAY APPC,SERVER

To investigate APPC work processed by APPC/MVS servers, enter the following command:

```
DISPLAY APPC,SERVER,ALL
```

This command produces output like the following:



```

ATB100I 14.27.48 APPC DISPLAY FRAME LAST F E SYS=MVS520
ALLOCATE QUEUES SERVERS QUEUED ALLOCATES
00002 00002 00000
STPN=LOOKUP
LLUN=Z01BAP03 PLUN=* USERID=*
PROFILE=* REGTIME=12/07/1995 19:05:56 QUEUED=00000
OLDEST=*NONE* LAST RCVD=00.21.02 TOT ALLOCS=00000001
SERVERS=00001 KEEP TIME=0000 TIME LEFT=*N/A*
ASNAME=RMILLER1
ASID=001C REGTIME=12/07/1995 19:05:56 TOT RCVD=00000001
RCVA ISS=19:05:56 RCVA RET=19:06:46
STPN=UPDATE
LLUN=Z01BAP03 PLUN=* USERID=*
PROFILE=* REGTIME=12/07/1995 19:04:25 QUEUED=00000
OLDEST=*NONE* LAST RCVD=*NONE* TOT ALLOCS=00000000
SERVERS=00001 KEEP TIME=0000 TIME LEFT=*N/A*
ASNAME=RMILLER2
ASID=000F REGTIME=12/07/1995 19:04:25 TOT RCVD=00000000
RCVA ISS=19:04:25 RCVA RET=19:04:25

```

Figure 110. Sample DISPLAY Output

According to the output, two APPC/MVS servers (shown by ASNAME) are running on the system: RMILLER1 and RMILLER2.

For RMILLER1, the output shows that the server:

- Has registered to serve allocate requests from the LOOKUP TP (shown by STPN=) that arrive on LU Z01BAP03.
- Has received 1 allocate request.

For RMILLER2, the output shows that the server:

- Registered to serve allocate requests from the UPDATE TP that arrive on LU Z01BAP03.
- Has not received any allocate requests for processing.

To view server processing from the perspective of the served TPs, enter the following command:

```
DISPLAY APPC,TP,ALL
```

This command produces output like the following:

```

ATB122I 14.26.53 APPC DISPLAY
LOCAL TP'S INBOUND CONVERSATIONS OUTBOUND CONVERSATIONS
00003 00001 00001
LTPN=UPDATE
LLUN=Z01BAP03 WUID=*UNKNOWN* CONVERSATIONS=00000 ASID=000F
SCHED=*NONE* ASNAME=RMILLER2 TPID=034ECC2800000001
STPN=LOOKUP
LLUN=Z01BAP03 WUID=*UNKNOWN* CONVERSATIONS=00001 ASID=001C
SCHED=*NONE* ASNAME=RMILLER1
PTPN=*UNKNOWN*
PLUN=USIBMZ0.Z01BAP01
PROTECTED=NO USERID=*NONE* DIRECTION=INBOUND
VERBS=00000001 IT=*NONE* LCID=034E2330
MODE=TRANPAR VTAMCID=*NONE* SYNC POINT IN PROG=NO
LUWID=*NONE*

```

Figure 111. Sample DISPLAY Output

Note that the TP ID field does not appear in the D APPC,TP,ALL output for RMILLER1 because RMILLER1 has received an inbound request. The TP ID appears for the server in address space RMILLER2 because it is not currently processing any allocate requests. Also, the TP running in address space RMILLER2 is also listed in the LTPN= field.

## Managing Use of the APPC/MVS API Trace Facility

---

Using the application programming interface (API) trace facility, your installation can collect data about APPC/MVS and CPI-C calls that an APPC/MVS TP issues. With this data, your installation can diagnose not only errors that occur during a specific call, but also problems with the conversation flow between the TP and its partners. This diagnostic capability is useful in both testing and production environments.

*z/OS MVS Programming: Writing Transaction Programs for APPC/MVS* contains the following information about using the API trace facility:

- Planning topics, such as possible security requirements for trace data sets; trace data set characteristics; and how to avoid losing trace data through wrapping or resource contention.
- How to select values for the ATBTRACE START request, to ensure that APPC/MVS collects the trace data required to either verify in TP processing, or diagnose errors.
- How to invoke the ATBTRACE REXX exec, including programming considerations, output, invocation methods, syntax diagrams, and parameter descriptions for each type of ATBTRACE request.
- General descriptions and examples of trace data set entries, with suggestions for sorting and reading the entries.

Because you might have to start, stop, or list tracing activity yourself, you should be familiar with all the information about the API trace facility in *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*.

### Planning for the Use of API Trace Data Sets

To use the API trace facility, programmers invoke the ATBTRACE REXX exec start, stop, or list tracing activity. Successful START and STOP requests require pre-allocated, sequential data sets to contain the trace entries and, depending on your installation's security policies, appropriate access to the data sets and APPC/MVS resources. APPC/MVS limits the number of API trace data sets per z/OS system to reduce the possible impact on performance.

Because of this limit and other contributing factors, such as the number of application programmers at your installation, you might have to decide how to set up the trace data sets for your installation. If so, read the information about trace data sets in *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*, which documents:

- Characteristics to select when you allocate the data sets
- Techniques for avoiding loss of data through wrapping
- Techniques for avoiding loss of data when APPC/MVS suspends tracing activity.

In general, a system programmer or administrator should have authority to start and stop tracing activity on any z/OS system. This ability is particularly advantageous for emergency situations in a production environment.

### Restoring API Tracing Capability

The API trace facility issues programmer messages to report:

- Successful and unsuccessful completion of an ATBTRACE START, STOP, or LIST request.
- Allocation or other problems with the data set specified on START and STOP requests.
- Timing or sequence problems (for example, delays in processing requests, START requests issued before STOP processing completes, and so on).
- Suspension or termination of API tracing activity by APPC/MVS. These conditions might result in the loss of trace data.

When APPC/MVS terminates API tracing activity, APPC/MVS issues message ATB499I to the operator. You cannot restore tracing capability without bringing down APPC/MVS itself. Because this process is so disruptive to workload, consider it only as a last resort. If tracing activity is absolutely critical on this

system, and you must restore the API trace facility by bringing down APPC/MVS and restarting it, make sure you follow these steps:

1. Use a SET APPC command to specify a parmlib member containing LUDEL statements for the APPC/MVS LUs
2. Allow the system some time to drain (or quiesce) APPC/MVS work
3. Only if necessary, use the methods in [“Using the VARY Command to Stop Work Immediately”](#) on page 191 to accelerate the process.

## Recovering from APPC problems

---

Advanced Program-to-Program Communications (APPC) attempts to restart automatically when it encounters an unrecoverable error. APPC contains two address spaces, APPC and the APPC/MVS transaction scheduler (ASCH). Both address spaces attempt to restart independently of each other so that, if the APPC/MVS transaction scheduler abnormally ends, APPC can continue normal processing.

For both address spaces you must perform recovery actions for the following:

- Initialization problems
- Abnormal ending with restart
- Abnormal ending without restart

### Recovery for the APPC Address Space

If the APPC address space detects an unrecoverable error, it attempts to restart itself. The system prevents continually recurring restarts as follows: If the APPC address space abnormally ends and attempts to restart twice in one hour, the system abnormally ends the APPC address space rather than restarting it again.

#### APPC Initialization Problems

If APPC cannot be initialized, the system issues message ATB008E. In this case, APPC will not perform properly if the operator enters a START APPC command.

#### Recovery Actions for APPC Initialization Problems

1. Report the problem to the system programmer.
2. When the problem is fixed, reIPL the system.

#### Abnormal End of the APPC address space with restart

If the system detects an unrecoverable error and abnormally ends the APPC address space, and APPC automatically restarts, message ATB005I indicates that APPC is restarting. In addition, the following occurs:

- The cross-system coupling facility (XCF) notifies the members of APPC's XCF group that APPC is unavailable.
- The system shuts down all active logical units (LU).
- The system does not save or restore active APPC conversations, either local and remote. The conversations will be lost and the system will notify the invoking applications.
- The system will not read the APPCPMxx parmlib member during restart. The parmlib members that were in effect prior to the restart will still be in effect following the restart.
- The system deletes the cache of TP profiles and side information. Following restart, the system refills the cache during normal processing.
- If APPC component trace is active before APPC abnormally ends, the system dumps the trace records when APPC abnormally ends and the APPC trace will not be active following restart.

- The system does not save or restore information regarding schedulers when APPC abnormally ends and restarts.
- XCF notifies the members of the APPC's XCF group that APPC is available once it restarts.

### **Recovery Actions for APPC Abnormal Ending with Restart**

1. Have applications redrive the system data file manager (SDFM) activate/deactivate TP profile interface to reset the TP profile information following the restart.
2. Restart APPC component trace and all APPC conversations that were active before the error when APPC services are again available.
3. Have schedulers re-identify themselves after APPC has restarted.

### **Abnormal end of the APPC address space without restart**

If the APPC address space ends due to either an operator CANCEL or FORCE command (messages ATB010I or ATB012I, respectively) or APPC/MVS cannot restart the APPC address space (message ATB006I), the following occurs:

- XCF notifies the members of APPC's XCF group that APPC is unavailable.
- The system shuts down all active LUs.
- The system does not save or restore active APPC conversations, either local and remote. The conversations are lost and the system notifies the invoking applications.
- FMH-5 requests that have been received, but not passed to the appropriate scheduler, are sent an FMH-7 notifying the partner TP of the error.

### **Recovery Actions for APPC Abnormal Ending without Restart**

1. Wait for the system to issue message ATB002I, indicating that the APPC address space has ended.
2. Enter a START APPC command to restart the APPC address space.

## **Recovery for the APPC/MVS Transaction Scheduler (ASCH) address space**

Like the APPC address space, if the APPC/MVS transaction scheduler (ASCH) detects an unrecoverable error, it attempts to restart itself. The system prevents continually recurring restarts as follows: If the APPC/MVS transaction scheduler abnormally ends and attempts to restart twice in one hour, the system abnormally ends the APPC/MVS transaction scheduler rather than restarting it again.

### **Abnormal Ending of the APPC/MVS Transaction Scheduler with Restart**

If the APPC/MVS transaction scheduler abnormally ends and automatically restarts, as indicated by message ASB050I, the following occurs:

- The system does not read the ASCHPMxx parmlib member during restart. The parmlib settings that were in effect prior to the restart are still in effect following the restart.
- The system marks all APPC initiators for deletion. The system deletes all waiting initiators immediately. The system deletes the initiators of active TPs as the TPs complete.

### **Recovery Actions for Abnormal Ending with Restart**

No recovery actions are necessary. The APPC/MVS transaction scheduler will restart automatically.

### **Abnormal Ending of the APPC/MVS Transaction Scheduler without Restart**

If the APPC/MVS transaction scheduler ends due to an operator CANCEL or FORCE command, or if it is unable to restart as indicated by message ASB051I, reason code X'00000001', the following occurs:

- The system marks all APPC initiators for deletion. The system deletes all waiting initiators immediately and deletes the initiators of active TPs as the TPs complete.

### **Recovery Actions for Abnormal Ending without Restart**

1. Wait for the system to issue message ASB053I, indicating that the APPC/MVS transaction scheduler address space has completed its ending process.
2. Enter a START ASCH command to restart the APPC/MVS transaction scheduler address space.



---

## Chapter 12. APPC/MVS Measurement and Tuning

APPC/MVS transaction programs can be identified as a separate type of work in an z/OS system. To integrate this work with existing work and to balance system resources, follow the suggestions provided in this chapter.

### References

This chapter assumes you are using Resource Measurement Facility (RMF) as the monitoring product. You could use an equivalent monitoring product instead of RMF.

*z/OS RMF Report Analysis*  
*z/OS MVS Initialization and Tuning Guide*  
*z/OS MVS Initialization and Tuning Reference*  
*z/OS MVS Installation Exits*  
*z/OS MVS System Commands*  
*z/OS MVS System Management Facilities (SMF)*  
*z/OS MVS System Messages, Vol 3 (ASB-BPX)*  
*z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*  
*z/OS Resource Measurement Facility User's Guide*

---

## Managing APPC Work in the System

Managing any work in the system requires a set of performance objectives in terms of resources used, response goals, and service requirements. To achieve these objectives, an installation monitors how work is running in the system and then tunes the system to try to meet the performance objectives.

Incorporating new work, such as APPC, into a tuned system can affect system performance. The purpose of this chapter is to provide guidance to help you get the best performance possible for APPC work without upsetting the balance of existing work in the system.

The steps for measuring the effect of APPC work in your system and for tuning the system are listed below. Each step is described in more detail in the sections that follow.

- 1. Consider the storage requirements of APPC/MVS**
- 2. Monitor APPC performance**
  - a. If resources used by TPs are charged to accounts, use SMF to audit APPC work.
  - b. Use RMF reports to measure the performance of APPC as a whole, by class, by account number, or by single transaction program.
  - c. Use the DISPLAY APPC and DISPLAY ASCH operator commands to take a "snapshot" of APPC work in the system.
- 3. Improve performance by program design and administration**
  - a. Make efficient use of callable services.
  - b. Be aware of keywords that cause performance degradation in the TP profile's JCL.
  - c. Consider using multi-trans scheduling to enhance performance.
  - d. Define classes for APPC work with appropriate response time goals.
  - e. Put each multi-trans TP in its own class.
  - f. Associate TPs to LUs with the appropriate level of access to eliminate unnecessary searching.
  - g. Eliminate unnecessary use of the TP message log.
- 4. Improve performance through system changes**

- a. Consider the amount of buffer storage needed for received data.
- b. Improve verification performance of RACF by using VLF. For complete information, see [z/OS Security Server RACF System Programmer's Guide](#).
- c. Eliminate unnecessary SMF recording.
- d. Improve network performance.
- e. Minimize use of APPC component tracing.

## Considering APPC/MVS Storage Requirements

---

Additional storage is used when processing APPC/MVS work. APPC/MVS requires storage for message buffers, internal control blocks, and in-storage copies of the TP profiles and side information. These storage structures are allocated in the APPC and ASCH address spaces, and additional data is kept in the MASTER address space. These three address spaces need 1-2 megabytes to start and run one of the sample transaction programs supplied with APPC/MVS. Executing a low rate of APPC/MVS transaction programs requires approximately 3 megabytes of processor storage in these address spaces. A large mainframe processor, such as an IBM 3090/300J executing only APPC/MVS transaction programs, requires approximately 12 megabytes of processor storage for the APPC and ASCH address spaces and the additional data kept in the MASTER address space. In all cases, you must also include the processor storage that the transaction programs require.

Idle APPC/MVS transaction initiators have approximately the same storage requirements as JES initiators. When these initiators are executing an APPC transaction program, the storage requirements depend on the requirements of the application.

Because of its need for increased processor storage, APPC/MVS may require additional auxiliary storage and paging space. For information about buffer storage limits, which is related to paging requirements, see [“Controlling Buffer Limit Size”](#) on page 216.

## Changing the Maximum for the System Workload

The MAXUSER parameter in IEASYSxx defines the total number of jobs and started tasks that can run concurrently in the system. To enable APPC work to run concurrently with existing work in the system, increase the MAXUSER value by the number of APPC initiators that will be running in the system. You can calculate the number of APPC initiators based on the type of APPC transaction scheduler classes and the number of initiators defined to those classes.

For example, assume your installation has three APPC/MVS transaction scheduler classes, A, B, and C. During peak production, classes A and B are likely to schedule their maximum number of initiators, but class C is not active until third shift when not much else is running. If class A has a maximum number of 25 initiators and class B has a maximum of 50, you would add 75 to the MAXUSER parameter to enable the APPC work to run with existing work during peak production.

## Monitoring APPC Performance

---

There are several ways you can define APPC work and monitor its status in the system. You can assign account numbers to transaction programs and to users of transaction programs for SMF auditing. You can view the measurable performance and accounting of APPC work in various RMF reports. In addition, the DISPLAY APPC and DISPLAY ASCH operator commands provide "snapshots" of LU configurations, scheduling classes, and the local and partner TPs in the system.

These techniques are discussed in the following sections.

## Using SMF to Audit APPC Work

For every job that issues APPC/MVS callable services, SMF writes summary conversation information in a type 30 record. This record reflects the total conversation activity for a particular job's address space. In addition, SMF writes a type 33, subtype 1 record specifically for APPC inbound work scheduled by the



APPC/MVS transaction scheduler, and a type 33, subtype 2 record for individual APPC/MVS conversations (inbound and outbound) on the system.

To use APPC conversation information and APPC transaction records, your installation must customize its accounting packages that read the SMF records.

## Information that SMF Records for APPC/MVS Work

The following table briefly summarizes the information that SMF provides for APPC/MVS.

Table 30. SMF Records for APPC/MVS		
SMF Record	When Written	Contents of Record
Type 30 (address space/time based)	For every job that uses APPC/MVS callable services	Routine type 30 information plus conversation information such as:  # of conversations # of send and receive calls Amount of data sent and received
Type 33, Subtype 1 (inbound transaction based)	For an inbound conversation scheduled by the APPC/MVS transaction scheduler and for each multi-trans Get_Transaction and Return_Transaction call	Conversation information similar to that written by the type 30 record, but specific to the transaction. TP information such as:  TP name Local and partner LU names User and account numbers I/O statistics  Scheduling information such as:  Schedule class Schedule type Specific dates and times work was received, queued, started, and ended.
Type 33, Subtype 2 (conversation based)	For all conversations on the system (inbound and outbound), including inbound conversations scheduled by the APPC/MVS transaction scheduler or processed by an APPC/MVS server, as well as all outbound conversations.	Detailed information about each conversation on the system, such as:  Conversation ID Name of the TP that issued the conversation request Local and partner LU name Number of sends and receives Amount of data sent and received.  When inbound conversations are processed by APPC/MVS servers, subtype 2 records also contain information about server processing, such as the specific dates and times that the conversation was:  Received by APPC/MVS Added to the server's allocate queue Received by the server for subsequent processing Deallocated.

A single APPC/MVS outbound or inbound transaction program generates at least one set of SMF type 30 records for job accounting. If the transaction program is inbound and is scheduled as standard, it generates the set of SMF type 30 records and a type 33 record (subtypes 1 and 2). If the transaction program is inbound and scheduled as multi-trans, it generates the set of SMF type 30 records for the TP as a whole, an SMF 33, subtype 1 record for shell initiation and another SMF type 33, subtype 1 record for shell termination, an SMF type 33 subtype 1 record for each Get\_Transaction and Return\_Transaction call, and an SMF type 33, subtype 2 record at the end of each conversation (when either partner program deallocates the conversation).

For more information about SMF records for APPC/MVS, see [z/OS MVS System Management Facilities \(SMF\)](#).

## Assigning Account Numbers to Transactions

The account number for inbound work comes from one of two places depending on whether the account is tailored. If the account is not tailored (the TP profile specifies `TAILOR_ACCOUNT(NO)`), the account number comes from the TP profile JCL JOB statement section as shown in the following example.

```
TPNAME(MIKES_TP)
TPSCHED_DELIMITER(+++)
CLASS(A)
TAILOR_ACCOUNT(NO)
JCL_DELIMITER(%%)
//MYTP      JOB  'MIKE' ...
//STEP1     EXEC PGM=MYTP ...

...
%0%0%
%0%0%
+++
```

Figure 112. Example of a TP Profile for No Account Tailoring

If the account is tailored (the TP profile specifies `TAILOR_ACCOUNT(YES)`), account numbers come from each user's security profile. Each new account number is retrieved from the `WORKATTR` segment of a RACF `ADDUSER` or `ALTUSER` command, as shown in one of the following examples.

```
TPNAME(MAIL)
TPSCHED_DELIMITER(+++)
CLASS(A)
TAILOR_ACCOUNT(YES)
JCL_DELIMITER(%%)
//MAIL      JOB  'DEPT5A' ...
//STEP1     EXEC PGM=MAILBAG ...

...
%0%0%
%0%0%
+++
```

Figure 113. Example of a TP Profile for Account Tailoring

```
ALTUSER USER22 WORKATTR (WANAME('LINDA MEYERS') WABLDG('BUILDING 9')
WADEPT('DEPARTMENT 5A') WAROOM('ROOM 22') WAADDR1('HOMEVILLE, NY')
WAACNT('22LMD5A'))
```

Figure 114. Example of a RACF User Profile for Account Tailoring

For information about RACF user profiles, see [Chapter 10, “Setting up Network Security,” on page 133](#). For information about how SMF audits APPC resources, see [z/OS MVS System Management Facilities \(SMF\)](#).

TPs that tailor account numbers can charge resources to a different account number for each instance of the TP. As account numbers change, an installation might want to validate the changed numbers. Exit IEFUAV allows for account validation before a new instance of a program runs. For information about IEFUAV, see [z/OS MVS Installation Exits](#).

## Using RMF Reports

RMF measures selected areas of system activity and presents information in printed reports or display reports.

To review the performance of APPC as a whole, by class, by account number, or by single transaction, you can request an RMF workload activity report. For a TP, you will need to provide the subsystem under

which the TP is running, which is ASCH by default. For information about the workload activity report and other reports that RMF generates, see [z/OS Resource Measurement Facility Report Analysis](#).

## Using the DISPLAY Operator Command

To see APPC configurations, issue variations of the DISPLAY APPC and DISPLAY ASCH commands as described below. These are particularly useful when you notice a problem or when people call the help desk with problems. Keep in mind, however, that if you repeatedly issue the DISPLAY command, you can degrade performance.

### DISPLAY APPC,TP

This command displays general TP status in terms of number of local TPs and numbers of inbound and outbound conversations. When you add parameters to the command, you get more detailed displays. The ALL parameter provides the most detail.

### DISPLAY APPC,LU

This variation of the DISPLAY command displays LU status, such as the number of active, outbound, pending, and terminating LUs. The most detailed information results when you add the ALL parameter.

### DISPLAY APPC,SERVER

This command displays information about APPC/MVS servers and the allocate queues they are serving. The most detailed information results when you add the ALL parameter.

### DISPLAY ASCH

The DISPLAY ASCH command displays the scheduling status for the APPC/MVS transaction scheduler. You can obtain the most detailed scheduling information with the ALL parameter.

For more information about the DISPLAY command, see Chapter 11, “Operating APPC/MVS,” on page 185. [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#) and [z/OS MVS System Commands](#) contain additional information.

## Improving Performance Through Program Design and Administration

---

The program design of a cooperative application for APPC/MVS can affect how efficiently it runs. After the TP is designed and coded, you can maximize its performance in the way you define it to the system. The following sections describe some of the programming and administrative steps you can take to improve the performance of APPC/MVS transaction programs.

### Making Efficient Use of Callable Services

The APPC/MVS callable services are not alike in terms of performance. For example, services that call VTAM or use I/O have a greater number of instructions than services that call an internal MVS function. Although using these services is often necessary, you can minimize their use or combine operations to be more efficient.

To see a table of performance considerations for individual APPC/MVS callable services and for a list of specific suggestions, see [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

### Avoiding Certain JCL Keywords

Using the following keywords in the JCL section of a TP profile affects performance:

- The system variable &SYSUID in a data set name causes performance degradation.

- JCLLIB statements also cause performance degradation.
- Using DISP=OLD for data sets accessible by more than one user can cause contention and degrade performance. If possible, use DISP=SHR instead.
- When using temporary data sets, avoid the use of &&DSNAME wherever possible. Specifying no data set name for temporary data sets allows the system to create a unique name and avoids contention between transaction programs for the same data set name.

## Using the Multi-Trans Schedule Type

The multi-trans schedule type can offer performance enhancements to TPs that are designed to take advantage of its scheduling benefits. For example, multi-trans TPs can do initiator start-up processing just once in the shell for the benefit of each multi-trans request. The multi-trans shell can also do some general cleanup processing when the initiator ends.

Another scheduling benefit occurs when multi-trans TPs allocate and open files in the shell. Thus, each request doesn't have to security check, allocate, open, and close each file.

Because the multi-trans shell remains active for only five minutes while waiting for a request, it is not recommended for applications where the anticipated rate of inbound requests is less than one every five minutes.

For more information about the multi-trans schedule type, see [“Multi-Trans Schedule Type” on page 30](#).

## Defining Classes and Response Time Goals

Defining APPC/MVS transaction initiator classes allows an installation to set scheduling algorithms for different types of transaction programs. Classes are defined in an ASCHPMxx parmlib member with the CLASSADD statement. Keywords in the CLASSADD statement define the minimum number of transaction initiators (MIN), the maximum number of transaction initiators (MAX), and the desired response time goal (RESPGOAL).

The APPC/MVS transaction scheduler starts and maintains at least the minimum number of transaction initiators for a class regardless of the amount of work in that class. Depending on the RESPGOAL, as more work comes in, more initiators may be started until the maximum is reached. Therefore you can control the number of initiators guaranteed to be available for a class and the upper limit of initiators available for the class. The APPC/MVS transaction scheduler, through calculating the response time goal you set for the class, determines the actual number of active initiators, using the MIN and MAX limits.

If you set MIN too high, system resources that may be needed elsewhere are left idle. If you set the MIN too low, the scheduler may waste time and resources creating and deleting transaction initiators. Similarly, if you set MAX too high, system resources that may be needed elsewhere are not available. If you set MAX too low, transaction programs will wait on the queue until an existing initiator becomes available, jeopardizing the response time goal for the class.

To optimize performance while still meeting resource goals, specify MIN and MAX values for transaction initiators that are percentages of the greatest number of anticipated transactions running in the class at any given time. You can determine the exact percentages after considering the number of transaction initiators available for all classes and experimenting with various values until one meets performance requirements.

The RESPGOAL is the total time in seconds that you want to allow for queueing and running a transaction program in a particular class. The APPC/MVS transaction scheduler begins calculating the response time from the moment the attach request for the transaction program enters the system. [Figure 115 on page 215](#) shows what the RESPGOAL keyword controls in terms of total end user response time.

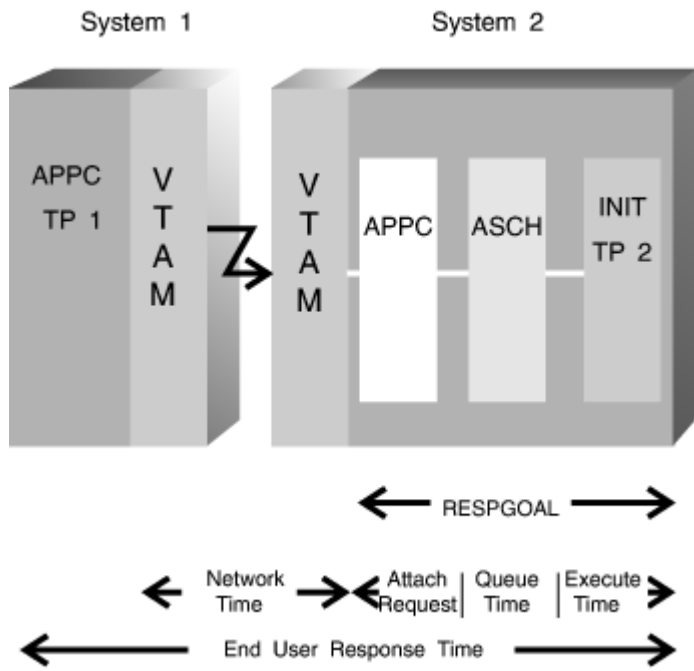


Figure 115. Response Time in an APPC/MVS Environment

To determine transaction run times and queue times, you can use RMF reports and SMF type 33 records. The APPC/MVS transaction scheduler can reduce the queue time by creating transaction initiators, but has no control over attach processing time and TP run time. Therefore, to set the RESPGOAL, determine the average run time for transactions in the class and add attach processing time and an allowable queue delay time. The additional queue delay time provides the APPC/MVS transaction scheduler with some control to attempt to optimize overall system overhead associated with creating and deleting transaction initiators.

## Putting Multi-Trans TPs in their Own Class

In general, it is a good idea to group TPs with similar characteristics in the same class. Therefore, place TPs with the same response time goals, with the same SYSOUT requirements, and with the same schedule type in the same class.

For multi-trans TPs, however, IBM recommends that each TP scheduled as multi-trans be assigned to a unique class of transaction initiators. If multi-trans TPs are mixed in a class with other multi-trans TPs or with TPs scheduled as standard, the minimum set of transaction initiators available for that class of TPs might change frequently depending on the type of transaction that needs to be processed. If a multi-trans TP takes a long time to initialize, other TPs in the class might be forced to wait for initiators.

## Associating TPs and LUs with the Appropriate Level

The use of levels for TP profiles and LUs allows an installation to control access to its TPs. Because the use of levels involves searching, you can reduce processing overhead and DASD I/Os by associating the appropriate level of TP with the appropriate LU.

A single TP can have profiles for three levels:

### System TP profile

One available for all users on the system (highest level)

### Group TP profile

One for a specified group of users

### User TP profile

One for an individual user (lowest level).

When APPC/MVS receives an incoming allocate request for a TP with more than one profile, it uses the TP profile with the lowest level to which the requestor has access.

Local LUs are also defined with a system, group, or user level to control the level of TP that runs in APPC/MVS.

#### **System**

An LU with a TP level of system allows only requests for system TP profiles. A request passing through this LU can access only system-level TP profiles.

#### **Group**

An LU with a TP level of group allows requests for group and system TP profiles. A request passing through this LU first attempts to access group-level TP profiles and then system-level TP profiles.

#### **User**

An LU with a TP level of user allows requests for user, group, and system TP profiles. A request passing through this LU first attempts to access user-level, then group-level, and finally system-level TP profiles.

The results of attempting to access a particular level of TP profile through an LU defined to a specific level are shown in Table 31 on page 216. The search starts at the lowest level of access and stops as soon as a match is found.

<i>Table 31. Local LU and TP profile level interactions</i>		
<b>TP Level of LU</b>	<b>Profiles Searched</b>	<b>Profiles Not Searched</b>
System	System TP profiles	Group and User TP profiles
Group	Group then System TP profiles	User TP profiles
User	User then Group then System TP profiles	None

To prevent unnecessary searching, assign system level TPs to system level LUs and avoid mixing system level TPs with user and group level TPs.

## **Limiting Use of the TP Message Log**

The TP message log can be used for recovery and problem determination when an error occurs while a TP is processing. You can control the TP message log through parameters from the TP profile and from the APPC/MVS transaction scheduler parmlib member (ASCHPMxx). Some parameters control how much I/O is required to write messages to the log, and how much storage APPC/MVS uses. Once a TP is in production, you might want to alter the parameter values to limit the use of logs. For a description of the parameters and possible values, see [“Logging Transaction Program Processing”](#) on page 33.

## **Improving Performance through System Changes**

A system programmer can make system changes to improve the overall performance of APPC. Some suggested changes follow.

## **Controlling Buffer Limit Size**

When a TP sends data to a partner TP on an z/OS system, the data is stored in buffers in the APPC address space until the partner TP receives it. When TPs send large amounts of data and their partners are slow to receive it, more and more buffers of data fill up virtual storage. This situation can cause a shortage of buffer space for other TPs and for the system itself.

APPC/MVS provides two functions that allow you to control the amount of buffer space that TPs use in the APPC address space:

- **Buffer size control:**

This function controls the amount of buffer storage in the APPC address space that is available to *all* TPs. Use this function to prevent buffer storage from becoming so large that it causes an auxiliary storage shortage when it is paged out, which can cause the system to go into a disabled wait.

- **Conversation-level pacing:**

This function controls the amount of buffer space that *any one conversation* can use at a particular time. Use this function when one or more conversations are using so much buffer space that they prevent other conversations from obtaining required buffer space. With conversation-level pacing, APPC/MVS allows data to flow into the buffer at a consistent rate, never allowing the data to exceed the buffer space limit for a conversation. APPC/MVS *paces* the conversation so it cannot use so much storage that it creates a shortage for other conversations. Pacing controls the rate of transmission of data to prevent overrun or congestion.

The following sections describe several methods that you can use to enable the functions listed above.

## Buffer Size Control

To control the buffer limit size, use the BUFSTOR parameter in the APPC start procedure in SYS1.PROCLIB. Specify the buffer storage limit in **megabytes**. Use one of the following methods:

- Specify the buffer storage limit of 88 megabytes used for MVS/ESA SP 4.2, if it is appropriate for your system. (The BUFSTOR parameter was not available in MVS/ESA SP 4.2, so systems were required to use the default buffer storage limit of 88 megabytes for that release.) Set BUFSTOR=88 in the PROC statement:

```
//      APPC PROC APPC=00, BUFSTOR=88
```

In the EXEC statement, set BUFSTOR=&BUFSTOR, as follows:

```
//      EXEC  PGM=ATBINITM, PARM=' APPC=&APPC, BUFSTOR=&BUFSTOR. . . '
```

If you use this method, you can specify BUFSTOR=xx on the START APPC command to change the BUFSTOR value.

- Define a fixed value for the BUFSTOR parameter, from 0 to 2048, on the EXEC statement. For example, if you want to set 48 megabytes as the buffer storage limit, specify the following:

```
//      EXEC  PGM=ATBINITM, . . . PARM=' APPC=&APPC, BUFSTOR=48 '
```

- Let APPC/MVS calculate a value based on your auxiliary storage. If you do not include the BUFSTOR parameter in the EXEC statement, APPC/MVS calculates a value that is approximately one third of the amount of free auxiliary storage your system has at the time APPC starts.

```
//      EXEC  PGM=ATBINITM, PARM=' APPC=&APPC. . . '
```

Each time the operator enters the START APPC command, the system re-calculates the buffer storage limit value.

When specifying the buffer storage limit on the BUFSTOR parameter, keep the following in mind:

- The value must be a decimal value between 0 and 2048. If you specify 0 or a value greater than 2048, the system sets the buffer storage limit to the maximum of 2048.
- If you specify a value less than or equal to 8, the system uses a value of 8. If you specify a value greater than 8, the system rounds the value down to the nearest multiple of 8.
- If you use a constant buffer storage limit value, you might want to calculate it from an RMF page/swap data set activity report taken at a peak period. Use numbers from the report in the following equations:

```
# available slots = (# slots allocated) - (max slots used)
MB of available storage = (# available slots)/256
```

Your buffer storage limit value should not exceed the MB of available storage that results from the second equation.

## Conversation Level Pacing

To control the amount of buffer space available to any one conversation, use the CONVBUFF parameter in the APPC start procedure in SYS1.PROCLIB. Specify the amount of buffer space in **kilobytes**. Use one of the following methods:

1. Specify a value for the for the CONVBUFF parameter on the PROC statement. For example, if you want to set 2000 kilobytes as the maximum amount of buffer space that any one conversation can use at one time, specify the following:

```
//      APPC  PROC  APPC=00,CONVBUFF=2000
```

In the EXEC statement, set CONVBUFF=&CONVBUFF, as follows:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC,CONVBUFF=&CONVBUFF...'
```

If you use this method, you can specify CONVBUFF=xx on the START APPC command to change the CONVBUFF value.

2. Define a fixed value for the CONVBUFF parameter on the EXEC statement. For example, if you want to set 2000 kilobytes as the maximum amount of buffer space that any one conversation can use at one time, specify the following:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC,CONVBUFF=2000...'
```

3. Let APPC/MVS use the default value for CONVBUFF, which is 1000 kilobytes. Code the exec statement as follows:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC...'
```

When specifying the amount of buffer space available to any one conversation on the CONVBUFF parameter, keep the following in mind:

- The maximum value you can specify is 2097152 kilobytes (decimal).
- If you do not specify a value, or if you specify a value of zero, APPC/MVS uses the default value of 1000 kilobytes.
- If you specify a value between 1 and 39 on the CONVBUFF parameter, the system uses a value of 40 (because APPC/MVS requires a minimum of 40 kilobytes of storage per conversation).
- If you specify a value that is not a multiple of four kilobytes (decimal), the system rounds the value of CONVBUFF up to the next highest multiple of four. For example, if you specify CONVBUFF=1023, the system makes 1024 kilobytes of buffer storage available to one conversation.
- If you specify a value on CONVBUFF that is greater than the buffer storage limit (which is either the default value calculated by APPC/MVS or the value specified on the BUFSTOR parameter), the system does the following:
  - Issues message ATB017I
  - Uses the buffer storage limit.

For example, the system performs the actions listed above if you enter the following command (which specifies a CONVBUFF value that is greater than the BUFSTOR value):

```
S  APPC, SUB=MSTR, BUFSTOR=8, CONVBUFF=2097148
```

## Minimizing Use of APPC Component Trace

APPC component trace, although beneficial for diagnostic information, degrades performance. To limit APPC tracing to abnormal events only, you can invoke component trace as follows:

```
TRACE CT,ON,COMP=SYSAPPC
```

When prompted for a reply, respond with:



```
REPLY id,END
```

Even this minimum amount of tracing can affect performance; to achieve maximum performance after APPC work is stable, turn APPC component trace off as follows:

```
TRACE CT,OFF,COMP=SYSAPPC
```

For more information about APPC component trace, see [z/OS MVS Diagnosis: Tools and Service Aids](#).

## Controlling SMF Type 33 Recording for APPC

Because each Get\_Transaction and Return\_Transaction call creates an SMF type 33 record, multi-trans programs can greatly increase the amount of data sent to the SMF data set. On the other hand, it is the type 33 record that contains the accounting information for APPC/MVS transaction programs.

If you need to conserve the amount of data sent to the SMF data set and do not need specific accounting for TPs scheduled as standard or multi-trans, turn off SMF recording for the SMF type 33 record. SMF recording is controlled by the TYPE and NOTYPE parameters in the SMFPRMxx parmlib member.

For more information about controlling SMF recording, see the SMFPRMxx description in [z/OS MVS Initialization and Tuning Reference](#).

## Improving Network Performance

Although network performance is difficult to control, here are suggestions for maximizing response time.

### Minimize Remote Communication Calls

Remote conversations almost always have longer response times than local conversations on a single system due to the communication calls that must pass from one system to another. The guidelines mentioned in [“Making Efficient Use of Callable Services” on page 213](#) apply especially to remote conversations. Therefore to improve response times, keep the network communication calls to a minimum and follow the list of suggestions from [z/OS MVS Programming: Writing Transaction Programs for APPC/MVS](#).

### Optimize LU-to-LU Sessions

To keep LU-to-LU sessions active during interruptions in either APPC/MVS or VTAM service, and to preserve new conversation requests until APPC/MVS service resumes, use VTAM persistent sessions. Single-node persistent sessions (SNPS) helps provide continuity of service to APPC/MVS TPs in situations where the APPC address space is cancelled, forced, terminated, or automatically restarted. The sessions also persist during interruptions in scheduler service. Multi-node persistent sessions (MNPS) extends persistence to sessions across VTAM failures as well and gives you the option to start the same LU on another system in the sysplex with all of its LU-to-LU sessions still intact.

If APPC/MVS or scheduler service is interrupted, new conversation requests targeted to LUs that were deactivated by the interruption will be queued until service returns or the PSTIMER time limit expires, whichever comes first. For MNPS, even if VTAM service is interrupted, VTAM will queue the new inbound conversation requests until the LU is reactivated on any system in the sysplex.

For an APPC/MVS LU that handles protected conversations, however, persistent sessions are not preserved if the LU fails while a syncpoint operation is in progress. Sessions on which syncpoint operations were in progress do not persist. All other sessions still have the persistent attribute. In this case, the session is unbound so that outstanding resynchronization work can proceed when the LU is reactivated.

With MNPS, persistent sessions go beyond the scope of just keeping sessions active when various services go down. An installation may choose to move a particular LU to another system in the sysplex. This can be done:

- to redistribute workload in the sysplex

- to move all work off of one system so that maintenance can be performed.
- to satisfy other availability issues.

Since the sessions associated with the LU move with the LU, this provides a non-disruptive mechanism to shift the current workload.

The PSTIMER parameter in the APPCPMxx LUADD statement controls whether persistent sessions are in effect for a particular local LU and how long the sessions persist. For more information about coding the PSTIMER parameter, see the APPCPMxx parmlib member description in [z/OS MVS Initialization and Tuning Reference](#).

SNPS support requires ACF/VTAM Version 3 Release 4. MNPS requires eNetwork Communications Server Release 5. For general information about persistent sessions, see [z/OS Communications Server: SNA Network Implementation Guide](#).

## Maximum number of APPC active conversations

Each APPC conversation on the system uses a small amount of APPC system resources in order to execute. If a single address space has a serious programmatic or logical error or if the workload volume is enormous, a tremendous number of active conversations within a single address space could result. Consequently, vast quantities of APPC system resources could be consumed, exhausting the ability of APPC to start up conversations in other address spaces. APPC on z/OS provides a mechanism which will notify an installation of such an event, and allow them to take the appropriate actions to fix this potentially crippling situation.

This function defines the maximum number of APPC active conversations that can be active in any particular address space at one time. The system either writes a critical action message to the console whenever an address space exceeds this maximum, or optionally, halts all new conversations within the address space when the limit has been reached.

To define the maximum number of active conversations, use the CONVMAX parameter in the APPC started procedure in specify the value for the maximum number of active conversations. Use one of the following methods:

1. Specify a value for the CONVMAX parameter on the PROC statement.

For example, if you want to set maximum of APPC active conversation as 3000, specify the following:

```
// APPC PROC APPC=00,CONVMAX=3000
```

In the exec statement set, CONVMAX=&CONVMAX as follows:

```
// EXEC PGM=ATBINITM,PARM="APPC=&APPC,CONVMAX=&CONVMAX...."
```

If you use this method, you can specify CONVMAX=xx on the START APPC command to change the CONVMAX value.

2. Define a fixed value for the CONVMAX parameter on the EXEC statement. For example, if you want to set 3000 as the maximum number of APPC active conversations, specify the following:

```
// EXEC PGM=ATBINITM,PARM="APPC=&APPC,CONVMAX=3000..."
```

3. Let APPC/MVS use the default value for CONVMAX, which is 2000. Code the exec statement as follows:

```
// EXEC PGM=ATBINITM,PARM="APPC=&APPC..."
```

When specifying the maximum number of APPC active conversations on the CONVMAX parameter, keep the following in mind:

- The value must be a decimal value between 100 and 20000. If you specify a value greater than 20000, the system sets the maximum number of active conversations threshold to 20000. If you specify a value between 1 and 99, the system sets the maximum APPC conversations threshold to 100.

- If you specify a value of 0 (zero) the system will not monitor the total number of active conversations for an address space, regardless of the quantity.

To define the action appc should take when the CONVMAX limit has been reached, use the CMACTION parameter in the APPC started procedure in SYS1.PROCLIB. There are two possible values for the CONVMAX action parameter:

- MSGONLY - cuts a critical action message whenever the number of active conversations in a single address space exceeds the CONVMAX value in effect on the system. APPC will not take any action on preventing new conversations from being started inside the affected address space.
- HALTNEW - prevents new conversations from being started inside an address space once the CONVMAX value has been reached. A critical action message is also sent to the console to inform the installation that no new conversations will be allowed to be started until the number of active conversations within the address space decreases. This is the default value.

Specify one of two possible values for the CMACTION parameter using one of the following methods:

1. Specify a value of the CMACTION parameter on the PROC statement to either MSGONLY or HALTNEW. For example, specify the following if you want APPC to halt all new conversations in an address space when the CONVMAX has been reached for that address space:

```
//      APPC  PROC  APPC=00,CMACTION=HALTNEW
```

Then, in the exec statement, set CMACTION=&CMACTION as follows:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC,
      CMACTION=&CMACTION...'
```

This method allow you to specify CMACTION=*cmaction\_value* on the START APPC command to change the CMACTION value.

2. Define a fixed value for the CMACTION parameter on the EXEC statement. For example, if you want only to be informed when the number of APPC active conversations has been exceeded, specify the following:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC,
      CMACTION=MSGONLY...'
```

3. Let APPC/MVS use the default value for CMACTION, which is HALTNEW. To do this, code the EXEC statement as follows:

```
//      EXEC  PGM=ATBINITM,PARM='APPC=&APPC...'
```



---

## Part 6. Installation checklists



## Chapter 13. Installing an APPC Application

When you install the MVS part of a cooperative application, you have different considerations depending on whether the MVS TP initiates an outbound conversation or responds to an inbound conversation request. In either situation, it is advisable to install the application first on a test system. After the application is running on the test system, you can add the security requirements. For information about security, see [Chapter 10, “Setting up Network Security,”](#) on page 133.

In addition to security considerations, you should also consider authorization. For the installation to work properly, your system should have an APF-authorized miglib data set in the LNKST concatenation. If necessary, refer to [z/OS MVS Initialization and Tuning Reference](#) for information about APF authorization for LNKST data sets.

This chapter describes the basic install requirements for an outbound request and for an inbound request; each description refers to other chapters for more detail.

SYS1.SAMPLIB contains examples showing how to install and run APPC applications. The examples are contained in the SYS1.SAMPLIB members whose names begin with ATBCA and ATBLA. See the ATBALL member of SYS1.SAMPLIB for descriptions of the examples.

### Installing a TP that Initiates an Outbound Request

The following steps are a suggested checklist for installing in MVS a TP that initiates an outbound request. Under each step is a specific example of how a programmer named Fred installs a cooperative application that is a distributed application between an z/OS server and an OS/2 workstation.

After the TP is loaded onto the host according to the directions supplied with the application, follow these steps to provide the support necessary to run the TP.

#### 1. Start APPC

1. Create an LU in an APPCPMxx parmlib member with an LUADD statement. (See [“Adding a Local LU — LUADD Statement”](#) on page 122.)

*Fred decides to take all the defaults on his test run. He codes a parmlib member named APPCPM10 and codes only the LU name on the LUADD statement. Fred's APPCPM10 parmlib member looks something like this:*

```
LUADD
ACBNAME(MVSTEST)
```

*Figure 116. Example*

2. Define a VSAM KSDS for side information. (See [“Defining the VSAM Key Sequenced Data Sets \(KSDS\)”](#) on page 56.)

*Fred has no idea how large to make the side information file, so he uses the sample definition he found in SYS1.SAMPLIB(ATBSIVSM).*

*Figure 117. Example*

3. Specify the VSAM KSDS name. (See [“Specifying a VSAM KSDS for Side Information — SIDEINFO Statement”](#) on page 126.)

Because Fred used the sample VSAM definition, the name of his VSAM KSDS is SYS1.APPCSI. This is also the default name in the SIDEINFO statement. To activate the default, the SIDEINFO statement must be coded. Fred adds the SIDEINFO statement to parmlib member APPCPM10, so the member now looks like this:

```
LUADD
  ACBNAME(MVSTEST)

SIDEINFO
```

Figure 118. Example

4. Code a VTAM APPL definition statement for the LU. (See “Defining the Local LU to VTAM” on page 105.)

Fred uses the example APPL statement in SYS1.SAMPLIB member ATBAPPL and changes both the APPL name and the name of the LU to MVSTEST.

Figure 119. Example

5. Define at least one LU 6.2 logon mode. (See “Defining an APPC Logon Mode” on page 104).

Because Fred's distributed application moves data from MVS to an OS/2 workstation, he needs a logon mode that controls pacing between unlike systems. He uses the sample for PC sessions (APPCPLM) found in Figure 64 on page 104 and gives it to the installation's VTAM programmer to add to the VTAM logon mode table.

Figure 120. Example

6. Depending on whether the APPC address space was started, do one of the following:

- If the APPC address space was not yet started, issue the START command. (See “Starting the APPC and ASCH Address Spaces” on page 185.)

From an operator's console, Fred issues the following START command:

```
START APPC,SUB=MSTR,APPC=10
```

Figure 121. Example

- If the APPC address space was already started, issue the SET command. (See “Dynamically Changing the APPC/MVS Environment” on page 187.)

From an operator's console, Fred issues the following SET command:

```
SET APPC=10
```

Figure 122. Example

## 2. Start ASCH

**Note:** Although outbound TPs do not necessarily use the APPC/MVS transaction scheduler, the ASCH address space must be started for APPC/MVS to work properly.

If the ASCH address space was not yet started, issue the START command. (See “Starting the APPC and ASCH Address Spaces” on page 185.)



From an operator's console, Fred issues the following START command:

```
START ASCH,SUB=MSTR
```

Figure 123. Example

### 3. Configure the peer system

For details, see information about the peer system.

1. Create a local LU.

**Note:** This local LU name is the partner LU name used in the MVS side information entry.

*Fred uses Communication Manager to create an APPC LU profile for a local LU named PS2TEST. He gives this LU name to his installation's VTAM programmer for use in the VTAM cross system configuration.*

Figure 124. Example

2. Define the MVS logon modes on the peer system.

*Logon modes are called transmission service modes in OS/2. Fred defines a transmission service mode for APPCPCLM, using the same name and the same parameter values for the OS/2 parameters as was used in the original VTAM definition.*

Figure 125. Example

3. Create a partner LU.

**Note:** This partner LU name must match the MVS local LU name.

*Using the MVS local LU name MVSTEST, Fred creates an APPC partner LU profile on OS/2. He adds the name of the transmission service mode to the profile.*

Figure 126. Example

4. Create a TP profile for the application according to the instructions that came with it and the requirements of the peer system.

*Fred creates a TP profile for the part of the distributed application that is installed on MVS. For specific values, he refers to the instructions that came with the application.*

Figure 127. Example

### 4. Create side information

1. Define a side information entry. (See [“Creating Side Information”](#) on page 65.)

*The part of Fred's distributed application that is installed on MVS is named Broadcast. Rather than create side information entries for all the workstations his program will communicate with, Fred creates a single entry for testing purposes. His entry looks like this:*

```
DESTNAME(TESTSITE)
MODENAME(APPCPCLM)
TPNAME(BROADCAST)
PARTNER_LU(PS2TEST)
```

*Figure 128. Example*

2. Add the side information entry to the side information file. (See Chapter 6, “Using the APPC/MVS Administration Utility,” on page 73 and Chapter 7, “Using the APPC/MVS Administration Dialog,” on page 83.)

*Fred chooses to use the APPC administration utility because the APPC administration dialog was not yet installed in TSO/E. He issues an SIADD based on the example he found in SYS1.SAMPLIB member ATBUTIL.*

*Figure 129. Example*

## Installing a TP that Responds to an Inbound Request

The following steps are a suggested checklist for installing a TP that responds to an inbound request. Under each step is a specific example of how a programmer named Jane installs a cooperative application that is a database accessing program between MVS and OS/2.

After the TP is loaded onto the system according to the directions supplied with the application, follow these steps to provide the support necessary to run the TP.

### 1. Start APPC

1. Create an LU in an APPCPMxx parmlib member with an LUADD statement. (See “Adding a Local LU — LUADD Statement” on page 122.)

*Jane decides to take all the defaults on her test run. She codes a parmlib member named APPCPM22 and only codes the LU name on the LUADD statement. Jane's APPCPM22 parmlib member looks something like this:*

```
LUADD
  ACBNAME(MVSLU)
```

*Figure 130. Example*

2. Define a VSAM KSDS for TP profile information. (See “Defining the VSAM Key Sequenced Data Sets (KSDS)” on page 56

*Jane has no idea how large to make the TP profile file, so she uses the sample definition she found in SYS1.SAMPLIB(ATBTPVSM).*

*Figure 131. Example*

3. Specify the VSAM KSDS name in the APPCPMxx LUADD statement following the TPDATA keyword parameter. (See “Adding a Local LU — LUADD Statement” on page 122.)

*Because Jane used the sample VSAM definition, the name of her VSAM KSDS is SYS1.APPCTP. This is also the default name in the LUADD statement. Therefore, Jane does not need to add anything to her parmlib member APPCPM22.*

*Figure 132. Example*

4. Code a VTAM APPL definition statement for the LU. (See [“Defining the Local LU to VTAM”](#) on page 105.)

*Jane uses the example APPL statement in SYS1.SAMPLIB member ATBAPPL, and changes both the APPL name and the name of the LU to MVSLU.*

*Figure 133. Example*

5. Depending on whether the APPC address space was started, do one of the following:

- If the APPC address space was not yet started, issue the START command. (See [“Starting the APPC and ASCH Address Spaces”](#) on page 185.)

*From an operator's console, Jane issues the following START command:*

```
START APPC, SUB=MSTR, APPC=22
```

*Figure 134. Example*

- If the APPC address space was already started, issue the SET command. (See [“Dynamically Changing the APPC/MVS Environment”](#) on page 187.)

*From an operator's console, Jane issues the following SET command:*

```
SET APPC=22
```

*Figure 135. Example*

## 2. Start ASCH

1. Create a class in an ASCHPMxx parmlib member with a CLASSADD statement. (See [“Defining a Class – CLASSADD Statement”](#) on page 45.)

*Jane's program is a multi-trans TP that she wants to run continuously in a class by itself. The expected response time goal for the TP is 1/2 second. Jane codes a parmlib member named ASCHPM11 and defines the class like this:*

```
CLASSADD
  CLASSNAME (DATA)
  MAX(3)
  MIN(1)
  RESPGOAL (.5)
  MSGLIMIT(1000)

  OPTIONS
    DEFAULT (DATA)
```

*Jane includes the OPTIONS statement with DATA as the default class.*

*Figure 136. Example*

2. Depending on whether the ASCH address space was started, do one of the following:

- If the ASCH address space was not yet started, issue the START command. (See [“Starting the APPC and ASCH Address Spaces”](#) on page 185.)

*From an operator's console, Jane issues the following START command:*

```
START ASCH,SUB=MSTR,ASCH=11
```

*Figure 137. Example*

- If the ASCH address space was already started, issue the SET command. (See [“Dynamically Changing the APPC/MVS Environment”](#) on page 187.)

*From an operator's console, Jane issues the following SET command:*

```
SET ASCH=11
```

*Figure 138. Example*

### 3. Configure the peer system

For details, see information about the peer system.

1. Create a local LU.

**Note:** This local LU name is an MVS partner LU name.

*Jane uses Communication Manager to create an APPC LU profile for a local LU named PS2LU. She gives this LU name to her installation's VTAM programmer for use in the VTAM cross system configuration.*

*Figure 139. Example*

2. Define the MVS logon modes on the peer system.

*Logon modes are called transmission service modes in OS/2. Jane defines a transmission service mode for APPCPCLM, using the same name and the same parameter values for the OS/2 parameters as was used in the original VTAM definition.*

*Figure 140. Example*

3. Create a partner LU.

**Note:** This partner LU name must match the MVS local LU name.

*Using the MVS local LU name MVSLU, Jane creates an APPC partner LU profile on the OS/2. She adds the name of the transmission service mode to the profile.*

*Figure 141. Example*

### 4. Create a TP profile

1. Define a TP profile entry. (See [“Creating a TP Profile”](#) on page 59.)

*Jane defines a TP profile entry that takes the defaults except for the parameters dealing with a multi-trans program. Her TP profile looks like this:*

```
TPNAME(TESTDATA)
TPSCHED_DELIMITER(##)
CLASS(DATA)
TPSCHED_TYPE(MULTI_TRANS)
GENERIC_ID(JANE)
JCL_DELIMETER(XX)
//TESTDATA JOB 'JANE',RWP,....
//STEP EXEC PGM=TESTDATA
//INPUT DD DSN=DATABASE.DATA,....
XX
##
```

*Figure 142. Example*

2. Add the TP profile to the TP profile file. (See Chapter 6, [“Using the APPC/MVS Administration Utility,”](#) on page 73 and Chapter 7, [“Using the APPC/MVS Administration Dialog,”](#) on page 83)

*Jane chooses to use the APPC administration utility because the APPC administration dialog was not yet installed in TSO/E. She issues a TPADD based on the multi-trans example she found in SYS1.SAMPLIB member ATBUTIL.*

*Figure 143. Example*



## Appendix A. Character Sets

APPC/MVS makes use of character strings composed of characters from one of the following character sets:

- Character set 01134, which is composed of the uppercase letters A through Z and numerals 0-9.
- Character set Type A, which is composed of the uppercase letters A through Z, numerals 0-9, national characters (@, \$, #), and must begin with either an alphabetic or a national character.
- Character set 00640, which is composed of the uppercase and lowercase letters A through Z, numerals 0-9, and 19 special characters. Note that APPC/MVS does not allow blanks in 00640 character strings.

These character sets, along with hexadecimal and graphic representations, are provided in the following table:

Table 32. Character Sets 01134, Type A, and 00640					
Hex Code	Graphic	Description	Character Set		
			01134	Type A	00640
40		Blank			
4B	.	Period			X
4C	<	Less than sign			X
4D	(	Left parenthesis			X
4E	+	Plus sign			X
50	&	Ampersand			X
5B	\$	Dollar sign		X (Note 1)	
5C	*	Asterisk			X (Note 2)
5D	)	Right parenthesis			X
5E	;	Semicolon			X
60	—	Dash			X
61	/	Slash			X
6B	,	Comma			X (Note 3)
6C	%	Percent sign			X
6D	_	Underscore			X
6E	>	Greater than sign			X
6F	?	Question mark			X
7A	:	Colon			X
7B	#	Pound sign		X (Note 1)	
7C	@	At sign		X (Note 1)	
7D	'	Single quote			X
7E	=	Equals sign			X
7F	"	Double quote			X
81	a	Lowercase a			X

*Table 32. Character Sets 01134, Type A, and 00640 (continued)*

Hex Code	Graphic	Description	Character Set		
			01134	Type A	00640
82	b	Lowercase b			X
83	c	Lowercase c			X
84	d	Lowercase d			X
85	e	Lowercase e			X
86	f	Lowercase f			X
87	g	Lowercase g			X
88	h	Lowercase h			X
89	i	Lowercase i			X
91	j	Lowercase j			X
92	k	Lowercase k			X
93	l	Lowercase l			X
94	m	Lowercase m			X
95	n	Lowercase n			X
96	o	Lowercase o			X
97	p	Lowercase p			X
98	q	Lowercase q			X
99	r	Lowercase r			X
A2	s	Lowercase s			X
A3	t	Lowercase t			X
A4	u	Lowercase u			X
A5	v	Lowercase v			X
A6	w	Lowercase w			X
A7	x	Lowercase x			X
A8	y	Lowercase y			X
A9	z	Lowercase z			X
C1	A	Uppercase A	X	X	X
C2	B	Uppercase B	X	X	X
C3	C	Uppercase C	X	X	X
C4	D	Uppercase D	X	X	X
C5	E	Uppercase E	X	X	X
C6	F	Uppercase F	X	X	X
C7	G	Uppercase G	X	X	X
C8	H	Uppercase H	X	X	X
C9	I	Uppercase I	X	X	X
D1	J	Uppercase J	X	X	X



Table 32. Character Sets 01134, Type A, and 00640 (continued)					
Hex Code	Graphic	Description	Character Set		
			01134	Type A	00640
D2	K	Uppercase K	X	X	X
D3	L	Uppercase L	X	X	X
D4	M	Uppercase M	X	X	X
D5	N	Uppercase N	X	X	X
D6	O	Uppercase O	X	X	X
D7	P	Uppercase P	X	X	X
D8	Q	Uppercase Q	X	X	X
D9	R	Uppercase R	X	X	X
E2	S	Uppercase S	X	X	X
E3	T	Uppercase T	X	X	X
E4	U	Uppercase U	X	X	X
E5	V	Uppercase V	X	X	X
E6	W	Uppercase W	X	X	X
E7	X	Uppercase X	X	X	X
E8	Y	Uppercase Y	X	X	X
E9	Z	Uppercase Z	X	X	X
F0	0	Zero	X	X	X
F1	1	One	X	X	X
F2	2	Two	X	X	X
F3	3	Three	X	X	X
F4	4	Four	X	X	X
F5	5	Five	X	X	X
F6	6	Six	X	X	X
F7	7	Seven	X	X	X
F8	8	Eight	X	X	X
F9	9	Nine	X	X	X

**Note:**

1. Avoid these characters because they display differently depending on the national language code page in use.
2. Avoid using the asterisk in TP names because it causes a subset list request when entered on panels of the APPC administration dialog and in DISPLAY APPC commands.
3. Avoid using the comma in TP names because it acts as a parameter delimiter when entered in DISPLAY APPC commands.



---

## Appendix B. Coding the APPCLOG Utility

APPCLOG is a regular load module that is packaged and shipped in the system data set SYS1.MIGLIB.

### Parameters

---

A set of input parameters read in through PARMS DD control the functions that the APPCLOG utility can perform. Each parameter has its own record. If a parameter is not specified, a default value will be used. The format of the parameter record is as follows:

KEYWORD=VALUE

The parameter keyword, immediately followed by the '=' character, is immediately followed by the parameter value with no spaces in-between. The keyword and its value can be either uppercase or lowercase.

For example, if you want APPCLOG to read the log stream from the youngest block to the oldest block, add the following parameter record to PARMS DD:

```
DIRECTION=YTO
```

The following are the input parameter keywords that APPCLOG understands and the default values accepted by these keywords:

#### HEXDUMP

The HEXDUMP keyword controls whether a hexadecimal dump of the PPLU is generated. This hexadecimal dump immediately follows the formatted PPLU dump.

The possible values for the HEXDUMP keyword are:

- NO - No hexadecimal dump is produced. This is the default value.
- YES - The hexadecimal dump is produced.

Example:

```
HEXDUMP=YES
```

#### STREAM

The STREAM keyword specifies the name of the log stream to read. If this parameter is not specified, the default value is:

```
ATBAPPC.LU.LOGNAMES
```

Example:

```
STREAM=APPCTEST.LOGSTREAM
```

#### DIRECTION

Log streams can be read from the youngest block ID to the oldest block ID or from the oldest to the youngest. The DIRECTION keyword specifies which direction APPCLOG reads the log stream.

The possible values for DIRECTION are:

- OTY - Read the log stream from the oldest block ID to the youngest block ID. This is the default value.
- YTO - Read the log stream from the youngest block ID to the oldest block ID.

Example:

```
DIRECTION=YTO
```

**MAX**

The MAX keyword controls the maximum number of blocks (PPLUs) to process. The value specified for the MAX keyword should be a decimal value of the number of blocks the user wants to dump and format. For example, if you want to process 17 PPLUs, specify:

```
MAX=17
```

The default value is to process all the blocks in the log stream.

**REPS**

The REPS keyword controls the number of times the log stream is read and dumped. The value specified for the REPS keyword should be a decimal number. For example, if you want to read the log stream 10 times, specify:

```
REPS=10
```

The default value for the REPS keyword is 1.

**SCRATCH**

The SCRATCH keyword controls whether the log stream scratch pad is dumped out.

The possible values for the SCRATCH keyword are:

- NO - The scratch pad is not dumped. This is the default value.
- YES - The scratch pad is dumped.

Example:

```
SCRATCH=YES
```

**ANAYLYZE**

The ANAYLYZE keyword controls whether the APPCLOG utility analyzes the log stream and scratch pad data for correctness and consistency.

The possible values for the ANAYLYZE keyword are:

- NO - No analysis is done on the log stream or scratch pad data. This is the default value.
- YES - Analysis is done on the log stream and scratch pad data. Results of the analysis are printed out along with the rest of the APPCLOG output.

Example:

```
ANAYLYZE=YES
```

**HELP**

The HELP keyword controls whether the help text for APPCLOG is printed out.

The possible values for the HELP keyword are:

- NO - No help text is printed. This is the default value.
- YES - The help text is printed. No other processing will be done.

Example:

```
HELP=YES
```

**DEBUG**

The DEBUG keyword controls whether diagnostic information is printed by APPCLOG when it is processing. The DEBUG should be used only to diagnose problems with the APPCLOG utility.

The possible values for the DEBUG keyword are:

- NO - No diagnostic information is printed. This is the default value.
- YES - Diagnostic information is printed.

Example:

```
DEBUG=YES
```

## Examples of using the APPCLOG Utility

```
//IXCAPPL JOB
//STEP1 EXEC PGM=APPCLOG
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//REPORT DD SYSOUT=*,DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//HELP DD DSN=SYS1.SAMPLIB(APPCLOGH),DISP=SHR
//PARMS DD *
DIRECTION=YTO
/*
```

Figure 144. Example of JCL to Use the APPCLOG Utility

## Sample output

APPCLOG can perform the following tasks:

- Read the log stream file that is generated and used by APPC.
- Dump and format the APPC PPLU entries and scratch pad entries
- Write these entries out to a file or spool file, depending on how the DD card in the JCL is coded.

The following example is the formatted entries for a PPLU and scratch pad:

## APPCLOG Formatted Dump

```
----- APPCLOG Input Params -----
DEBUG=YES
SCRATCH=YES
ANALYZE=YES
HEXDUMP=YES

----- APPC Logstream Report -----
DEBUG: Issuing IXGCONN request=CONNECT
Dumping Log Stream:          ATBAPPC.LU.LOGNAMES

DEBUG: Issuing IXGBRWSE request=START OLDEST...
DEBUG: Reading Logstream ...
DEBUG: Direction=OLDTOYOUNG...
DEBUG: Browse Service Elapsed 00:00:00.000251
>>> PPLU ENTRY 00000001
      PPLU Field Name      Address      Value
-----
Atbpplu                    00006298  ATBPPLU
Version number             000062A0  01
FLink - Forward link       000062A4  00000000
BLink - Backward link      000062A8  00000000
Partner LU Name            000062AC  USIBMZ0.Z0A0AP04
Synpt capabilities         000062BF  6C
Log name                   000062C0  ATR.C451337F19628DA3.IBM
Log name Length            00006300  00000018
Flags                     00006305  00010000
  Xln Status               00006305  Xln_Complete, '01'X
  Init_Restart_Resync      00006306  OFF
  Block_IsA_Duplicate      00006306  OFF
  Initiating Plna         00006306  OFF
Latch                     00006308  Set Token = 7F38EC00000000BD
                               Set Number = 0000001C
Local Xln Latch            00006314  Set Token = 7F38EC00000000BD
                               Set Number = 0000001A
Remote Xln Latch           00006320  Set Token = 7F38EC00000000BD
                               Set Number = 0000001B
ResyncCount                0000632C  00000000
```

```

Valid Block Id      00006338 00000000 00000000
ReWritten Block Id  00006340 00000000 00000000
LDeleted Block Id   00006348 00000000 00000000
Chained Block Id    00006350 00000000 00000000
ResyncLatch         00006358 Set Token = 7F38EC00000000BD
                        Set Number = 0000001D

Local LU Name       00006364 USIBMZ0.Z097AP03
Partner ptr         00006378 0082F828
Correlator Name     0000637C USIBMZ0.Z0A0AP04

-- Hex Dump of PPLU --
+00000000 C1E3C2D7 D7D3E440 F0F10000 00000000 *ATBPPLU 01 *
+00000010 00000000 E4E2C9C2 D4E9F04B E9F0C1F0 * USIBMZ0.Z0A0*
+00000020 C1D7F0F4 4000006C C1E3D94B C3F4F5F1 *AP04 %ATR.C451*
+00000030 F3F3F7C6 F1F9F6F2 F8C4C1F3 4BC9C2D4 *337F19628DA3.IBM*
+00000040 40404040 40404040 40404040 40404040 * *
+00000050 40404040 40404040 40404040 40404040 * *
+00000060 40404040 40404040 00000018 00010000 * *
+00000070 7F38EC00 000000BD 0000001C 7F38EC00 *"Ö " "Ö *
+00000080 000000BD 0000001A 7F38EC00 000000BD * " " "Ö *
+00000090 0000001B 00000000 00000000 00000000 * ? *
+000000A0 00000000 00000000 00000000 00000000 * *
+000000B0 00000000 00000000 00000000 00000000 * *
+000000C0 7F38EC00 000000BD 0000001D E4E2C9C2 *"Ö " USIB*
+000000D0 D4E9F04B E9F0F9F7 C1D7F0F3 40000000 *MZ0.Z097AP03 *
+000000E0 0082F828 E4E2C9C2 D4E9F04B E9F0C1F0 * b8USIBMZ0.Z0A0*
+000000F0 C1D7F0F4 40000001 *AP04 *

DEBUG: Reading Logstream ...
DEBUG: Direction=OLDTOYOUNG...
DEBUG: Browse Service Elapsed 00:00:00.000319
>>> PPLU ENTRY 00000002

PPLU Field Name      Address      Value
-----
Atbplu               00006398 ATBPPLU
Version number        000063A0 01
FLink - Forward link 000063A4 00000000
BLink - Backward link 000063A8 00000000
Partner LU Name       000063AC USIBMZ0.Z097AP03
Synpt capabilities    000063BF 6C
Log name              000063C0 ATR.C451337F19628DA3.IBM
Log name Length       00006400 00000018
Flags                 00006405 00010000
  Xln Status          00006405 Xln_Complete, '01'X
  Init_Restart_Resync 00006406 OFF
  Block_IsA_Duplicate 00006406 OFF
  Initiating Plna     00006406 OFF
Latch                 00006408 Set Token = 7F38EC00000000BD
                        Set Number = 00000020
Local Xln Latch       00006414 Set Token = 7F38EC00000000BD
                        Set Number = 0000001E
Remote Xln Latch      00006420 Set Token = 7F38EC00000000BD
                        Set Number = 0000001F
ResyncCount           0000642C 00000000
Valid Block Id        00006438 00000000 00000000
ReWritten Block Id    00006440 00000000 00000000
LDeleted Block Id     00006448 00000000 00000000
Chained Block Id      00006450 00000000 00000000
ResyncLatch           00006458 Set Token = 7F38EC00000000BD
                        Set Number = 00000021
Local LU Name         00006464 USIBMZ0.Z0A0AP04
Partner ptr           00006478 0082F88C
Correlator Name       0000647C USIBMZ0.Z097AP03

-- Hex Dump of PPLU --
+00000000 C1E3C2D7 D7D3E440 F0F10000 00000000 *ATBPPLU 01 *
+00000010 00000000 E4E2C9C2 D4E9F04B E9F0F9F7 * USIBMZ0.Z097*
+00000020 C1D7F0F3 4000006C C1E3D94B C3F4F5F1 *AP03 %ATR.C451*
+00000030 F3F3F7C6 F1F9F6F2 F8C4C1F3 4BC9C2D4 *337F19628DA3.IBM*
+00000040 40404040 40404040 40404040 40404040 * *
+00000050 40404040 40404040 40404040 40404040 * *
+00000060 40404040 40404040 00000018 00010000 * *
+00000070 7F38EC00 000000BD 00000020 7F38EC00 *"Ö " "Ö *
+00000080 000000BD 0000001E 7F38EC00 000000BD * " " "Ö *
+00000090 0000001F 00000000 00000000 00000000 * *
+000000A0 00000000 00000000 00000000 00000000 * *
+000000B0 00000000 00000000 00000000 00000000 * *
+000000C0 7F38EC00 000000BD 00000021 E4E2C9C2 *"Ö " ?USIB*
+000000D0 D4E9F04B E9F0C1F0 C1D7F0F4 40000000 *MZ0.Z0A0AP04 *
+000000E0 0082F88C E4E2C9C2 D4E9F04B E9F0F9F7 * b8USIBMZ0.Z097*
+000000F0 C1D7F0F3 40000002 *AP03 *

DEBUG: Reading Logstream ...
DEBUG: Direction=OLDTOYOUNG...
DEBUG: Browse Service Elapsed 00:00:00.000317
>>> PPLU ENTRY 00000003

```

PPLU Field Name	Address	Value
Atbpplu	00006498	ATBPPLU
Version number	000064A0	01
FLink - Forward link	000064A4	00000000
BLink - Backward link	000064A8	7EEDABA0
Partner LU Name	000064AC	USIBMZ0.Z0B0AP04
Synpt capabilities	000064BF	6C
Log name	000064C0	ATR.C451337F19628DA3.IBM
Log name Length	00006500	00000018
Flags	00006505	00010000
Xln Status	00006505	Xln_Complete, '01'X
Init_Restart_Resync	00006506	OFF
Block IsA Duplicate	00006506	OFF
Initiating Plna	00006506	OFF
Latch	00006508	Set Token = 7F38EC00000000BD Set Number = 00000024
Local Xln Latch	00006514	Set Token = 7F38EC00000000BD Set Number = 00000022
Remote Xln Latch	00006520	Set Token = 7F38EC00000000BD Set Number = 00000023
ResyncCount	0000652C	00000000
Valid Block Id	00006538	00000000 00000000
ReWritten Block Id	00006540	00000000 00000000
LDeleted Block Id	00006548	00000000 00000000
Chained Block Id	00006550	00000000 00000001
ResyncLatch	00006558	Set Token = 7F38EC00000000BD Set Number = 00000025
Local LU Name	00006564	USIBMZ0.Z097AP03
Partner ptr	00006578	0082F8F0
Correlator Name	0000657C	USIBMZ0.Z0B0AP04
-- Hex Dump of PPLU --		
+00000000	C1E3C2D7 D7D3E440 F0F10000 00000000	*ATBPPLU 01 *
+00000010	7EEDABA0 E4E2C9C2 D4E9F04B E9F0C2F0	* USIBMZ0.Z0B0*
+00000020	C1D7F0F4 4000006C C1E3D94B C3F4F5F1	*AP04 %ATR.C451*
+00000030	F3F3F7C6 F1F9F6F2 F8C4C1F3 4BC9C2D4	*337F19628DA3.IBM*
+00000040	40404040 40404040 40404040 40404040	* *
+00000050	40404040 40404040 40404040 40404040	* *
+00000060	40404040 40404040 00000018 00010000	* *
+00000070	7F38EC00 000000BD 00000024 7F38EC00	*"ö " "ö *
+00000080	000000BD 00000022 7F38EC00 000000BD	* " "ö "**
+00000090	00000023 00000000 00000000 00000000	* *
+000000A0	00000000 00000000 00000000 00000000	* *
+000000B0	00000000 00000000 00000000 00000001	* *
+000000C0	7F38EC00 000000BD 00000025 E4E2C9C2	*"ö " *
USIB*		
+000000D0	D4E9F04B E9F0F9F7 C1D7F0F3 40000000	*MZ0.Z097AP03 *
+000000E0	0082F8F0 E4E2C9C2 D4E9F04B E9F0C2F0	* b80USIBMZ0.Z0B0*
+000000F0	C1D7F0F4 40000003	*AP04 *
DEBUG: Reading Logstream ...		
DEBUG: Direction=OLDTOYOUNG...		
DEBUG: Browse Service Elapsed 00:00:00.000557		
>>> PPLU ENTRY 00000004		
PPLU Field Name	Address	Value
Atbpplu	00006598	ATBPPLU
Version number	000065A0	01
FLink - Forward link	000065A4	00000000
BLink - Backward link	000065A8	00000000
Partner LU Name	000065AC	USIBMZ0.Z097AP03
Synpt capabilities	000065BF	6C
Log name	000065C0	ATR.C451337F19628DA3.IBM
Log name Length	00006600	00000018
Flags	00006605	00010000
Xln Status	00006605	Xln_Complete, '01'X
Init_Restart_Resync	00006606	OFF
Block IsA Duplicate	00006606	OFF
Initiating Plna	00006606	OFF
Latch	00006608	Set Token = 7F38EC00000000BD Set Number = 00000028
Local Xln Latch	00006614	Set Token = 7F38EC00000000BD Set Number = 00000026
Remote Xln Latch	00006620	Set Token = 7F38EC00000000BD Set Number = 00000027
ResyncCount	0000662C	00000000
Valid Block Id	00006638	00000000 00000000
ReWritten Block Id	00006640	00000000 00000000
LDeleted Block Id	00006648	00000000 00000000
Chained Block Id	00006650	00000000 00000000
ResyncLatch	00006658	Set Token = 7F38EC00000000BD Set Number = 00000029
Local LU Name	00006664	USIBMZ0.Z0B0AP04

```

Partner ptr          00006678 0082F954
Correlator Name      0000667C USIBMZ0.Z097AP03
-- Hex Dump of PPLU --
+00000000 C1E3C2D7 D7D3E440 F0F10000 00000000 *ATBPPLU 01 *
+00000010 00000000 E4E2C9C2 D4E9F04B E9F0F9F7 * USIBMZ0.Z097*
+00000020 C1D7F0F3 4000006C C1E3D94B C3F4F5F1 *AP03 %ATR.C451*
+00000030 F3F3F7C6 F1F9F6F2 F8C4C1F3 4BC9C2D4 *337F19628DA3.IBM*
+00000040 40404040 40404040 40404040 40404040 * *
+00000050 40404040 40404040 40404040 40404040 * *
+00000060 40404040 40404040 00000018 00010000 * *
+00000070 7F38EC00 000000BD 00000028 7F38EC00 *"Ö " "Ö *
+00000080 000000BD 00000026 7F38EC00 000000BD * " "Ö "*"
+00000090 00000027 00000000 00000000 00000000 * *
+000000A0 00000000 00000000 00000000 00000000 * *
+000000B0 00000000 00000000 00000000 00000000 * *
+000000C0 7F38EC00 000000BD 00000029 E4E2C9C2 *"Ö " USIB*
+000000D0 D4E9F04B E9F0C2F0 C1D7F0F4 40000000 *MZ0.Z0B0AP04 *
+000000E0 0082F954 E4E2C9C2 D4E9F04B E9F0F9F7 * b9èUSIBMZ0.Z097*
+000000F0 C1D7F0F3 40000004 *AP03 *
DEBUG: Reading Logstream ...
DEBUG: Direction=OLDTOYOUNG...
DEBUG: Browse Service Elapsed 00:00:00.003549
End of Logstream
Dumping Scratch Pad: Start
SCRATCH PAD CREATED SUCCESSFULLY
Scratch Pad Field Name Address Value
-----
ScratchPad_Id 00015000 ATBSCRPD
OldestBlockId 00015008 00000000 00000000
RewrittenBlockId 00015010 00000000 00000000
LocalLuEntries 00015018 0000
LogDelBlockEntries 0001501A 0000
Dumping Scratch Pad: End
DEBUG: Issuing IXGCONN request=DISCONNECT...

```

## APPCLOG Analysis Dump

The following is an example of the analysis report APPCLOG generates when the ANAYLYZE=YES parm is specified. In this example, APPCLOG has discovered the errors described in the list below:

- PPLU number 1 has an invalid eye catcher.
- The version number for PPLU number 1 is not correct.
- The fields Logically Deleted BlockId and Rewritten Block Id in PPLU number 1 have incompatible values.
- The field Partner LU Name in PPLU number 1 contains an invalid character.
- The field Logname in PPLU number 1 contains an invalid character.
- The field Local LU Name in PPLU number 1 contains an invalid character.
- The field Correlator Name in PPLU number 1 contains an invalid character.
- PPLU number 1 and PPLU number 2 are duplicates, meaning that both have the same values in their Partner LU Name and Local LU Name fields.
- PPLU number 2 and PPLU number 1 are duplicates, a repeat of the error found earlier.
- The Scratch Pad has and invalid eye catcher.

**Note:** The examples of the Formatted dump and Analysis dump were created from different log streams.

```

----- APPCLOG Analyze Data -----
PPLU number 00000001 has an eyecatcher which is not 'ATBPPLU '
PPLU number 00000001 has a version number which is not 01
PPLU number 00000001
  Has a Logically Deleted BlockId: 00000000 00000020
  and the Rewritten BlockId: 00000000 00000010
  They cannot both be non-zero
PPLU number 00000001 the field 'Partner LU Name '
  has an invalid character in postion 00000003
PPLU number 00000001 the field 'Logname '
  has an invalid character in postion 00000006
PPLU number 00000001 the field 'Local LU Name '
  has an invalid character in postion 00000010
PPLU number 00000001 the field 'Correlator Name '

```



```
has an invalid character in postion 00000005
PPLU number 00000001 is a duplicate of PPLU entry 00000002
PPLU number 00000002
  Has a Logically Deleted BlockId: 00000000 00000020
  and the Rewritten BlockId: 00000000 00000010
  They cannot both be non-zero
PPLU number 00000002 is a duplicate of PPLU entry 00000001
The Scratch Pad has an eyecatcher which is not 'ATBSCRPD'
```

To see the actual invalid data refer, see [“ APPCLOG Formatted Dump” on page 239](#) and find the appropriate PPLU and field. For example, the error message

```
PPLU number 00000001 has an eyecatcher which is not 'ATBPPLU '
```

would be generated if you saw the following eye catcher value.

```
>>> PPLU ENTRY 00000001
      PPLU Field Name    Address      Value
-----
Atbpplu                00006298 ATBPXXX
```



---

## Appendix C. Accessibility

Accessible publications for this product are offered through [IBM Documentation for z/OS \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS documentation see [How to Send Feedback to IBM](#) to leave documentation feedback.



## Notices

---

This information was developed for products and services that are offered in the USA or elsewhere.

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, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan*

**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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

*IBM Corporation  
Site Counsel  
2455 South Road*

Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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.

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

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Terms and conditions for product documentation

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## **Rights**

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## **IBM Online Privacy Statement**

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at [ibm.com/privacy](http://ibm.com/privacy) and IBM's Online Privacy Statement at [ibm.com/privacy/details](http://ibm.com/privacy/details) in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at [ibm.com/software/info/product-privacy](http://ibm.com/software/info/product-privacy).

## **Policy for unsupported hardware**

---

Various z/OS elements, such as DFSMSdfp, JES2, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## **Minimum supported hardware**

---

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

## Programming Interface Information

---

This document primarily documents information that is NOT intended to be used as a Programming Interface of z/OS.

This document also documents information that is intended to be used as a Programming Interface. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

Programming Interface Information
-----------------------------------

End Programming Interface Information
---------------------------------------

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

Microsoft, Windows, Windows NT, 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.

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.

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







Product Number: 5655-ZOS

SA23-1388-70

