

z/OS
2.5

*MVS Programming:
Writing Transaction Programs
for APPC/MVS*



Note

Before using this information and the product it supports, read the information in [“Notices” on page 481.](#)

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2021-09-30

© **Copyright International Business Machines Corporation 1991, 2021.**

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

Contents

Figures.....	ix
Tables.....	xiii
About This Document.....	xvii
Who Should Use This Document.....	xvii
How to Use This Document.....	xvii
How to Read Syntax Diagrams.....	xvii
Where to Find More Information.....	xviii
Related documents.....	xviii
z/OS information.....	xix
How to send your comments to IBM.....	xxi
If you have a technical problem.....	xxi
Summary of changes.....	xxiii
Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 5 (V2R5).....	xxiii
Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 4 (V2R4).....	xxiii
Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 3 (V2R3).....	xxiii
Part 1. Introduction.....	1
Chapter 1. Introduction to APPC/MVS.....	3
APPC Overview.....	3
APPC Concepts and Commonly Used Terms.....	5
What is APPC/MVS?.....	9
Overview of an APPC/MVS Outbound Request.....	13
Overview of an APPC/MVS Inbound Request.....	14
Steps to Write and Install an APPC/MVS Transaction Program.....	16
Part 2. Programming.....	19
Chapter 2. Designing and Writing an APPC/MVS Transaction Program.....	21
Benefits of APPC/MVS for Application Programs.....	21
The Elements of Conversation.....	22
Conversation States.....	23
APPC Conversation Services.....	23
Identifying TP Partners to MVS.....	24
Relating MVS Callable Services to CPI Communications.....	25
Flow Diagrams of Typical APPC/MVS Conversations.....	26
TP Environment and Design Considerations.....	29
The General APPC/MVS Environment for Transaction Programs.....	30
Portability and MVS-Specific Services.....	32
Security.....	34
Using Basic or Mapped Conversations.....	34
Data Conversion.....	35

Using Protected Conversations.....	35
Error Handling and Deallocation of Conversations.....	36
Chapter 3. Using CPI Communications.....	37
CPI Communications in APPC/MVS.....	37
Invocation Details for CPI Communications.....	38
Interface Definition Files (IDFs) for CPI-C Calls.....	38
Transaction Program (TP) Environment.....	39
Calling CPI Communications Routines.....	40
Parameter Descriptions.....	41
Required Modules.....	41
Conversation States.....	41
Performance Considerations.....	41
Chapter 4. The APPC/MVS Programming Interface.....	45
APPC/MVS TP Conversation Services.....	45
APPC/MVS TP Conversation States.....	45
Guide to the Conversation Services.....	46
Setting a Timeout Value for Potential Network Delays.....	55
Performance Considerations for Conversation Services.....	56
Advanced TP Services.....	58
Extracting Detailed Scheduling and Conversation Information.....	58
Adding User Data to Accounting Records.....	59
Using TP Schedule Types.....	59
Identifying and Deallocating Conversations with Outstanding Asynchronous Requests.....	64
Rejecting Conversations.....	65
Testing TPs.....	65
System Services.....	65
Example APPC/MVS Transaction Programs.....	65
Chapter 5. Installing and Testing Transaction Programs.....	67
Installing a TP for Testing.....	67
Testing a TP on MVS.....	67
Methods You Can Use to Create a Test Shell.....	67
Descriptions of APPC/MVS Test Services.....	68
Test Shell Characteristics.....	68
Calling APPC/MVS Test Services from Your Application.....	69
Using the TSO/E TEST Command to Test an Assembler Language TP.....	70
Testing a TP under APPC/MVS Scheduling.....	72
Requesting a User-Level or Group-Level TP Profile.....	72
Requesting Access to a User-Level TP Profile.....	73
Requesting Side Information.....	73
Enabling an LU for User-Level TP Profiles.....	73
Collecting Problem Data for Errors that Occur During Testing.....	73
Displaying APPC Activity on MVS.....	74
Tracing APPC Conversations.....	74
Putting a Tested TP into Production.....	74
Replacing an Active TP.....	75
Chapter 6. Diagnosing Problems with APPC/MVS TPs.....	77
Comparing the Detectives: Error_Extract, API Trace, and the TP Message Log.....	78
Clues: What Information They Collect.....	79
Modus Operandi: How They Interrogate Suspects.....	79
Fees: How to Reduce the Cost of the Investigation.....	79
The Initial Consultation: Building Your Crime Lab.....	80
The All-Star Collaboration: A Team Approach.....	80
Calls that Error_Extract or API Trace Support.....	81
Diagnosing TP Conversation Errors with the API Trace Facility.....	82

Setting Up API Trace Data Sets.....	83
Starting API Tracing Activity.....	86
Using the ATBTRACE REXX Exec.....	93
Interpreting API Trace Data.....	103
Overview of Error_Extract Service.....	108
Types of Error Information that Error_Extract Returns.....	108
Rules for Calling Error_Extract.....	109
Calling Error_Extract for an Unestablished Conversation.....	110
Using Error_Extract for Synchronous and Asynchronous Calls.....	110
Diagnosing Product-Specific Errors.....	114
Part 3. Reference.....	117
Chapter 7. Invocation Details for APPC/MVS Callable Services.....	119
APPC/MVS Program Environment.....	119
High-Level Language Compilers.....	120
Syntax and Linkage Conventions for the Callable Services.....	120
Parameter Description for Callable Services.....	121
Required Modules.....	122
Versions of Callable Services.....	122
Interface Definition Files (IDFs) for LU6.2 and APPC/MVS Services.....	122
Chapter 8. APPC/MVS TP Conversation Callable Services.....	125
Allocate.....	125
Confirm.....	140
Confirmed.....	147
Deallocate.....	150
Error_Extract.....	158
Flush.....	163
Get_Attributes.....	165
Get_Conversation.....	171
Get_TP_Properties.....	175
Get_Type.....	179
Post_on_Receipt.....	181
Prepare_to_Receive.....	185
Receive_Immediate.....	196
Receive_and_Wait.....	208
Request_to_Send.....	224
Send_Data.....	227
Send_Error.....	236
Set_Syncpt_Options.....	248
Set_TimeOut_Value.....	254
Chapter 9. APPC/MVS Advanced TP Callable Services.....	259
Advanced TP Callable Services with Multiple Call Names.....	259
Asynchronous_Manager.....	259
Accept_Test.....	262
Cleanup_TP.....	263
Extract_Information.....	266
Get_Transaction.....	272
Register_Test.....	274
Reject_Conversation.....	277
Return_Transaction.....	281
Set_Conversation_Accounting_Information.....	282
Unregister_Test.....	285
Version_Service.....	287

Chapter 10. API Trace Facility Messages.....	291
Chapter 11. Error_Extract Reason Codes and Messages.....	313
Summary of Error_Extract Reason Codes.....	313
Error_Extract Error Log Information (ASB, ATB7) Messages.....	317
Error_Extract (ATB8) Messages.....	343
Appendix A. Character Sets.....	387
Appendix B. Explanations of Return Codes for CPI Communications Services.....	391
Appendix C. APPC/MVS Conversation State Table.....	399
Explanation of State-Table Abbreviations.....	399
Conversation Characteristics ().....	399
Return Code Values [].....	400
Data_received and Status_received {, }.....	402
Table Symbols.....	403
How to Use the State Table.....	403
Appendix D. Support for SNA LU 6.2 Verbs and Option Sets.....	419
Mapping APPC/MVS TP Services to LU 6.2 Verbs and CPI Communications.....	419
APPC/MVS Support for LU 6.2 Option Sets.....	420
Flush the LU's Send Buffer (101).....	420
Get Attributes (102).....	420
Prepare to Receive (105).....	420
Receive Immediate (106).....	420
Sync Point Services (108).....	421
Get Conversation Type (110).....	421
Queued Allocation of a Conwinner Session (201).....	421
Immediate Allocation of a Session (203).....	421
Conversations between Programs Located at the Same LU (204).....	421
Session-Level LU-LU Verification (211).....	421
User ID Verification (212).....	421
Program Supplied User ID and Password (213).....	421
User ID Authorization (214).....	422
Profile Verification and Authorization (215).....	422
Origin LU Authorization (216).....	422
Profile Passthrough (217).....	422
Program-Supplied Profile (218).....	422
Receive Persistent Verification (220).....	422
Receive SIGNON/Change Password (222).....	422
Accounting (243).....	422
Long Locks (244).....	423
Test for Request-to-Send Received (245).....	423
Vote Read-Only Response to a Sync Point Operation (249).....	423
Extract Transaction and Conversation Identification Information (251).....	423
CHANGE_SESSION_LIMIT Verb (501).....	423
Session-Level Mandatory Cryptography (611).....	423
Appendix E. Previous Versions of APPC/MVS Callable Services.....	425
ATBALLC - Allocate (For MVS/ESA 4.2 and 4.2.2).....	425
ATBALC2 - Allocate (For MVS/ESA 4.3 through OS/390 Release 7).....	433
ATBALC5 - Allocate (For OS/390 Release 8 through z/OS V1R6).....	448
ATBCMCTU - Cleanup_TP (Unauthorized, for MVS/ESA 4.2).....	463
ATBGTA2 - Get_Attributes (For MVS/ESA 4.3 through z/OS V1R6).....	465
ATBGETP - Get_TP_Properties (For MVS/ESA 4.2 through OS/390 V1R2).....	471

ATBST05 - Set_TimeOut_Value (For OS/390 Release 8 through z/OS V1R6).....	474
Appendix F. Accessibility.....	477
Accessibility features.....	477
Consult assistive technologies.....	477
Keyboard navigation of the user interface.....	477
Dotted decimal syntax diagrams.....	477
Notices.....	481
Terms and conditions for product documentation.....	482
IBM Online Privacy Statement.....	483
Policy for unsupported hardware.....	483
Minimum supported hardware.....	483
Programming Interface Information.....	484
Trademarks.....	484
Index.....	485

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).....	14
12. APPC/MVS Communication Services (Inbound).....	15
13. One-Way Conversation.....	26
14. Two-Way Conversation.....	27
15. Example of a Confirmed Transaction.....	28
16. Example of Send_Error in Receive State.....	29
17. Example Use of the Conversation Correlator.....	49
18. Obtaining Asynchronous Notification With Post_on_Receipt.....	54
19. Obtaining Asynchronous Notification With Receive_and_Wait.....	54
20. Standard Scheduling for an APPC/MVS Transaction Program.....	60
21. Multi-Trans Scheduling for an APPC/MVS Transaction Program.....	61
22. Phases of Multi-Trans Processing.....	62
23. Use of Test Services by a TP Test Shell.....	70

24. TSO/E Terminal Screen.....	71
25. Sample Installation Configuration.....	87
26. Timing the Start of API Tracing Activity.....	88
27. Using VTAM Generic Resource Names.....	89
28. Using Symbolic Destination Names.....	89
29. Collecting API Trace Data for Concurrent Conversations.....	90
30. Collecting API Trace Data for Concurrent Conversations.....	91
31. Collecting API Trace Data for TPs with Multiple Levels.....	92
32. Example Use of Error_Extract Service, Synchronous.....	111
33. Example Use of Error_Extract Service, Asynchronous (figure continued).....	112
34. Example Use of Error_Extract Service, Asynchronous.....	113
35. ATBALC6 - LU 6.2 Allocate.....	126
36. ATBCFM - LU 6.2 Confirm.....	140
37. ATBCFMD - LU 6.2 Confirmed.....	148
38. ATBDEAL - LU 6.2 Deallocate.....	151
39. ATBEES3 - LU 6.2 Error_Extract.....	159
40. ATBFLUS - LU 6.2 Flush.....	163
41. ATBGTA6 —LU 6.2 Get_Attributes.....	166
42. ATBGETC - Get_Conversation.....	172
43. ATBGTP4 - LU 6.2 Get_TP_Properties.....	176
44. ATBGETT - LU 6.2 Get_Type.....	180
45. ATBPOR2 - Post_on_Receipt.....	183
46. ATBPTR - LU 6.2 Prepare_to_Receive.....	186
47. ATBRCVI - LU 6.2 Receive_Immediate.....	197
48. ATBRCVW - LU 6.2 Receive and Wait.....	209

49. ATBRTS - LU 6.2 Request to Send.....	225
50. ATBSEND - LU 6.2 Send Data.....	228
51. ATBSERR - LU 6.2 Send Error.....	237
52. ATBSSO4 - LU 6.2 Set_Syncpt_Options.....	249
53. ATBSTO6 - Set_TimeOut_Value.....	255
54. ATBAMR1 - Asynchronous_Manager.....	260
55. Invocation of the Accept_Test Callable Service.....	262
56. ATBCUC1 - Cleanup_TP (Unauthorized Version).....	264
57. ATBEXAI - Information Extract Service.....	267
58. ATBGTRN - Obtaining the Next Transaction.....	273
59. Invocation of the Register_Test Callable Service.....	275
60. ATBRJC2 - Reject_Conversation.....	277
61. ATBTRN - Restoring the Generic Environment.....	281
62. ATBSCA2 - Set_Conversation_Accounting_Information.....	283
63. Invocation of the Unregister_Test Callable Service.....	286
64. ATBVERS - Callable Service Version Service.....	288
65. ATBALLC - LU 6.2 Allocate.....	426
66. ATBALC2 - LU 6.2 Allocate.....	435
67. ATBALC5 - LU 6.2 Allocate.....	449
68. ATBCMCTU - Cleanup_TP (Unauthorized Version).....	463
69. ATBGTA2 - LU 6.2 Get Attributes.....	466
70. ATBGETP - LU 6.2 Get_TP_Properties.....	472
71. ATBSTO5 - Set_TimeOut_Value.....	475

Tables

1. Mapping of MVS TP Services and CPI Communications.....	25
2. IDFs Provided by APPC/MVS for CPI-C Calls.....	39
3. Some High-Level Language Compilers for CPI Communications Calls on z/OS.....	40
4. Performance Considerations for APPC/MVS CPI Communications Calls.....	42
5. Performance Considerations for MVS TP Callable Services.....	56
6. Where to Get Examples of APPC/MVS Interfaces.....	65
7. List of APPC/MVS-Related Materials in SYS1.MACLIB.....	66
8. Methods For Testing TPs on APPC/MVS.....	68
9. Selecting a Diagnostic Tool.....	77
10. Calls that Error_Extract or API Trace Support.....	81
11. Types of Information that Error_Extract Returns.....	108
12. Reason Codes for Product-Specific Errors.....	114
13. Some High-Level Language Compilers for APPC/MVS Calls.....	120
14. IDFs for APPC/MVS Conversation Calls.....	123
15. IDFs for APPC/MVS Allocate Queue Services.....	123
16. TP Conversation Callable Services with Multiple Call Names.....	125
17. Local LUs for Which an Address Space Can Allocate.....	133
18. Default Local LUs Used If None Are Specified.....	134
19. Return Codes for the Allocate Service.....	137
20. Return Codes for the Confirm Service.....	142
21. Return Codes for the Confirmed Service.....	149
22. Return Codes for the Deallocate Service.....	154
23. Return Codes for the Flush Service.....	164

24. Return Codes for the Get_Attributes Service.....	171
25. Return Codes for the Get_Conversation Service.....	174
26. Return Codes for the Get_TP_Properties Service.....	179
27. Return Codes for the Get_Type Service.....	181
28. Return Codes for the Post_on_Receipt Service.....	184
29. Return Codes for the Prepare_to_Receive Service.....	190
30. Return Codes for the Receive_Immediate Service.....	201
31. Return Codes for the Receive_and_Wait Service.....	218
32. Return Codes for the Request_to_Send Service.....	226
33. Return Codes for the Send_Data Service.....	231
34. Return Codes for the Send_Error Service.....	242
35. Return and Reason Codes for Set_Syncpt_Options.....	251
36. Advanced TP Callable Services with Multiple Call Names.....	259
37. Return and Reason Codes for Reject_Conversation.....	279
38. Return and Reason Codes for Set_Conversation_Accounting_Information.....	284
39. Example Values Returned by the Version_Service.....	287
40. Character Sets 01134, Type A, and 00640.....	387
41. Return Codes for CPI Communications Services.....	391
42. States and Transitions for APPC/MVS Conversation Calls.....	406
43. List of APPC/MVS TP Callable Services with SNA and CPI-C Equivalents.....	419
44. Local LUs for Which an Address Space Can Allocate.....	442
45. Default Local LUs Used If None Are Specified.....	443
46. Return Codes for the Allocate Service.....	445
47. Local LUs for Which an Address Space Can Allocate.....	455
48. Default Local LUs Used If None Are Specified.....	457

49. Return Codes for the Allocate Service.....	459
50. Return Codes for the Get_Attributes Service.....	470
51. Return Codes for the Get_TP_Properties Service.....	473

About This Document

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* (TPs), together form cooperative processing applications that can exploit the strengths of different computer architectures. This document tells how to design and write APPC transaction programs to run on MVS.

In this document, the term *APPC/MVS 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.

Who Should Use This Document

This document is for application programmers who design and write APPC/MVS transaction programs.

How to Use This Document

This document is one of the set of programming documents for MVS. This set describes how to write programs in high-level languages, such as C, FORTRAN, and COBOL. For more information about the content of this set of documents, see [z/OS Information Roadmap](#).

How to Read Syntax Diagrams

Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 contains syntax diagrams for the ATBTRACE REXX exec, which allows application programmers to control tracing activity for their TPs. Use the following instructions to learn how to read these diagrams.

Syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The ►— symbol indicates the beginning of a macro.

The —► symbol indicates that the macro syntax is continued on the next line.

The ►— symbol indicates that a macro is continued from the previous line.

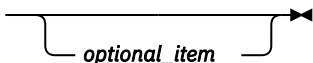
The —►◄ symbol indicates the end of a macro.

An *italicized* lower-case word indicates a variable.

- Required items appear on the horizontal line (the main path).

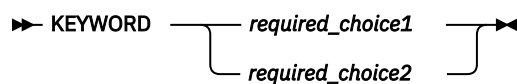
►► KEYWORD — *required_item* —►◄

- Optional items appear below the main path.

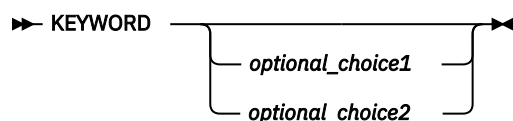
►► KEYWORD —  —►◄

- If you can choose from two or more items, they appear vertically, in a stack.

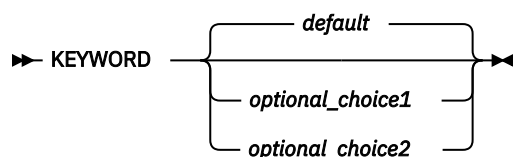
If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



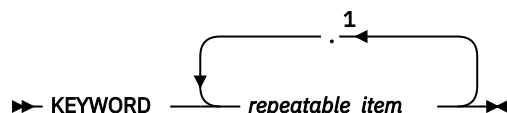
If one of the items has a default, it appears above the main path and the overriding choices will be shown below the line.



- An arrow returning to the left above the main line indicates an item that can be repeated indefinitely.



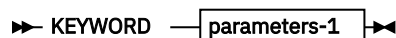
A repeat arrow with a syntax note indicates how many times this can be repeated.



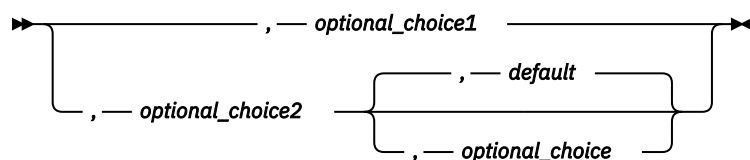
Notes:

¹ Specify the <parameter> 1 to n times.

- The $\left| \text{parameters-}n \right|$ symbol indicates a labelled group that continues below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.



parameters-1



Where to Find More Information

Related documents

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

Short Title Used in This document	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>VM/ESA Connectivity Planning, Administration, and Operation</i>	<i>VM/ESA Connectivity Planning, Administration, and Operation</i>	SC24-5448

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xxi.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function

If your comment or question is about z/OS itself, submit a request through the [IBM RFE Community](http://www.ibm.com/developerworks/rfe/) (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function

If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content

If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS MVS Writing TPs for APPC/MVS, SA23-1397-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the [IBM Support Portal](http://support.ibm.com) (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

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 \(www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument\)](http://www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 5 (V2R5)

The following content is new, changed, or no longer included in V2R5.

New

The following content is new.

- None

Changed

The following content is changed.

- None

Deleted

The following content was deleted.

- None

Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 4 (V2R4)

This information contains no technical changes for this release.

Summary of changes for z/OS MVS Programming: Writing Transaction Programs for APPC/MVS for Version 2 Release 3 (V2R3)

This information contains no technical changes for this release.

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.

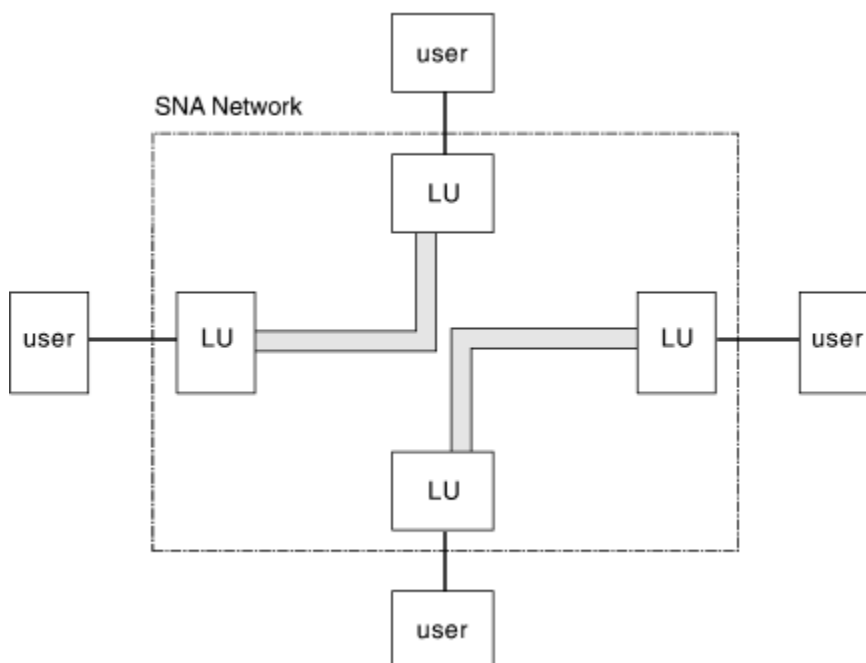


Figure 1. Network Communications between LUs and Users

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.

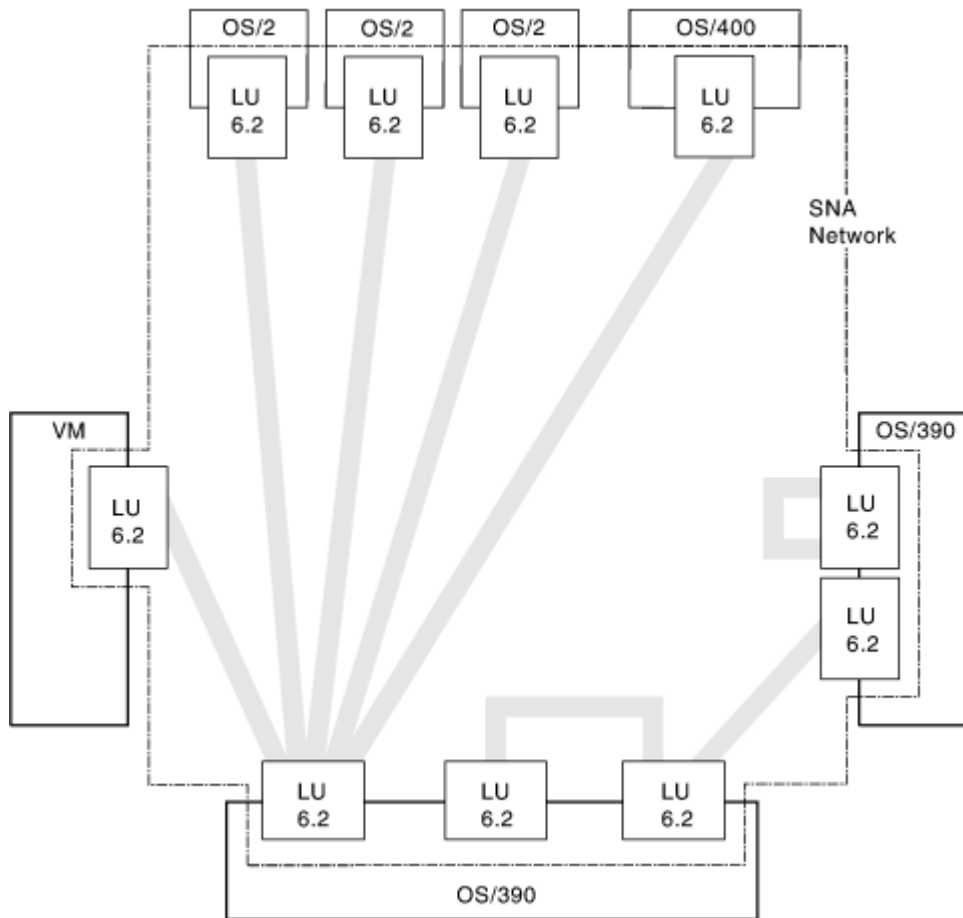


Figure 2. An SNA Network for Communications between Different Systems

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.

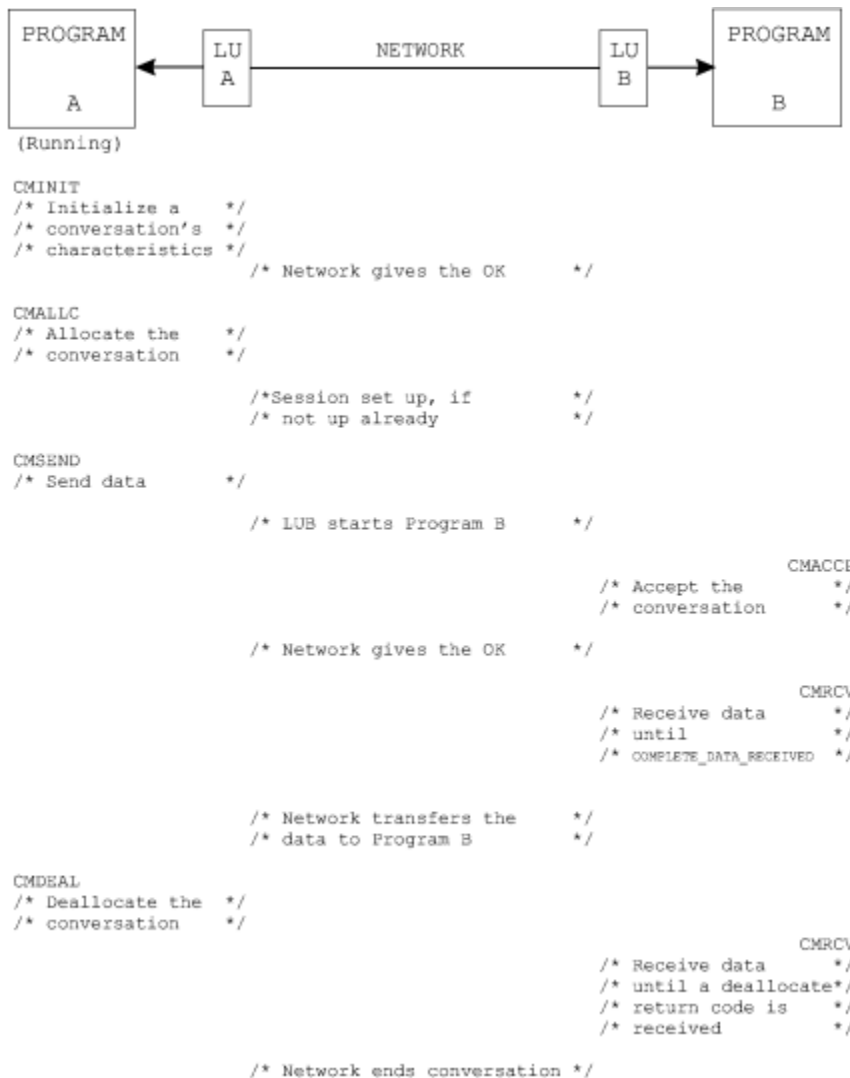


Figure 3. CPI Communications Program Scenario

This book tells how to write such APPC applications using both CPI Communications and a set of callable services that is specific to APPC/MVS.

APPC Concepts and Commonly Used Terms

Before writing APPC applications, 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.

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.

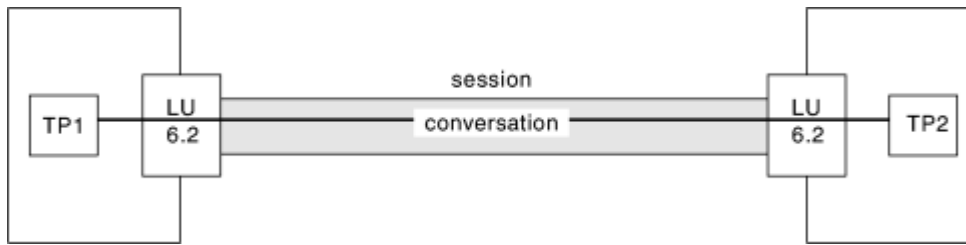


Figure 5. A Conversation between Two TPs

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.

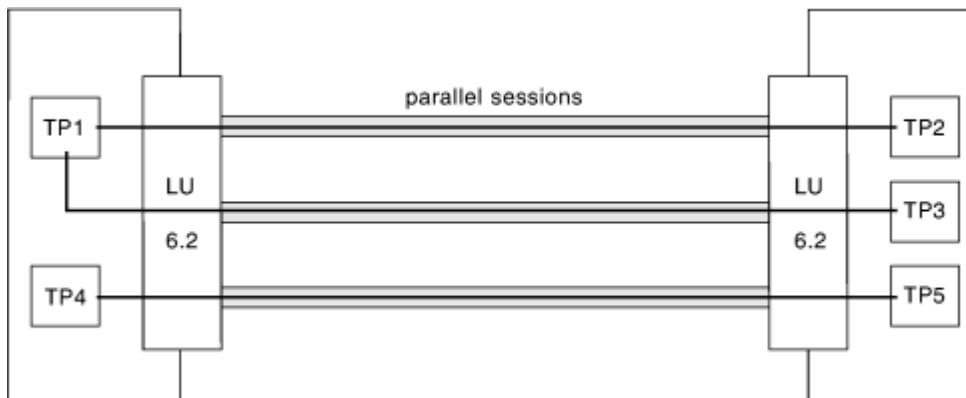


Figure 6. Parallel Sessions between LUs

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.

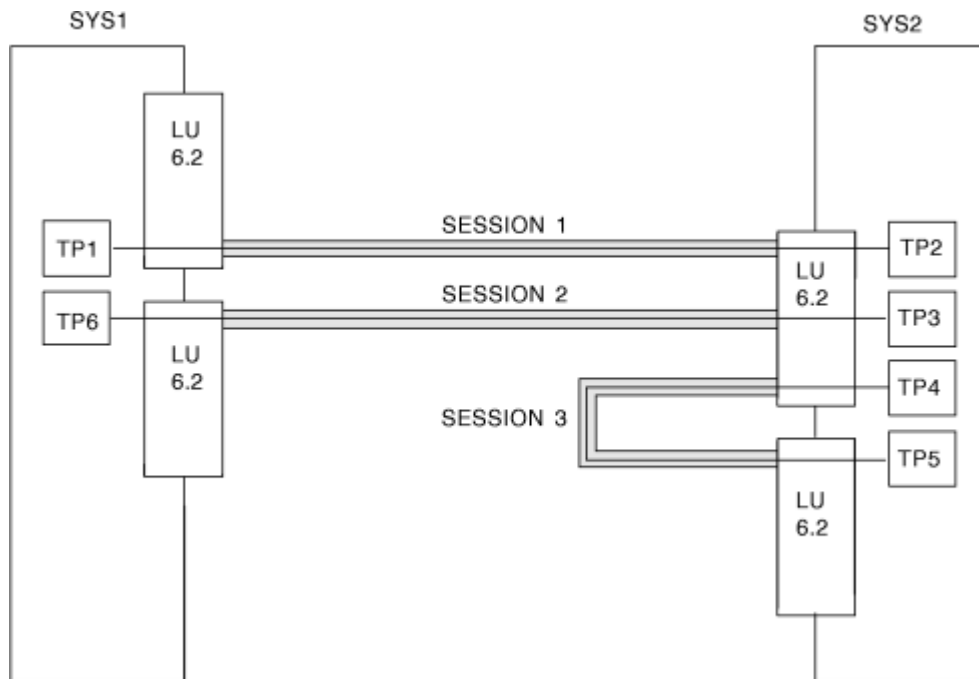


Figure 7. Different Types of Sessions between Two LUs

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.

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*.

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.

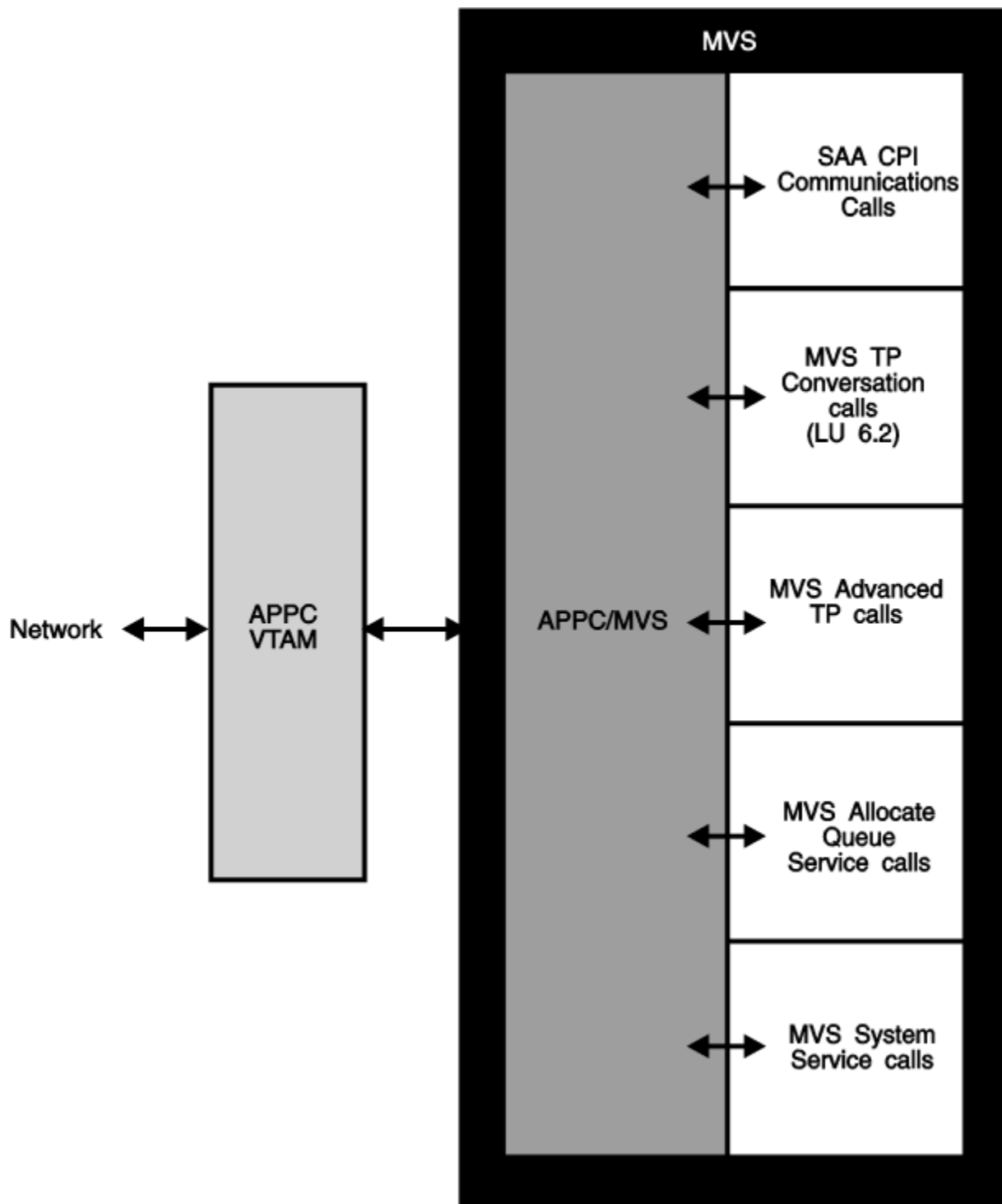


Figure 8. Types of APPC/MVS Callable Services

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* Chapter 3, “Using CPI Communications,” on page 37.

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.

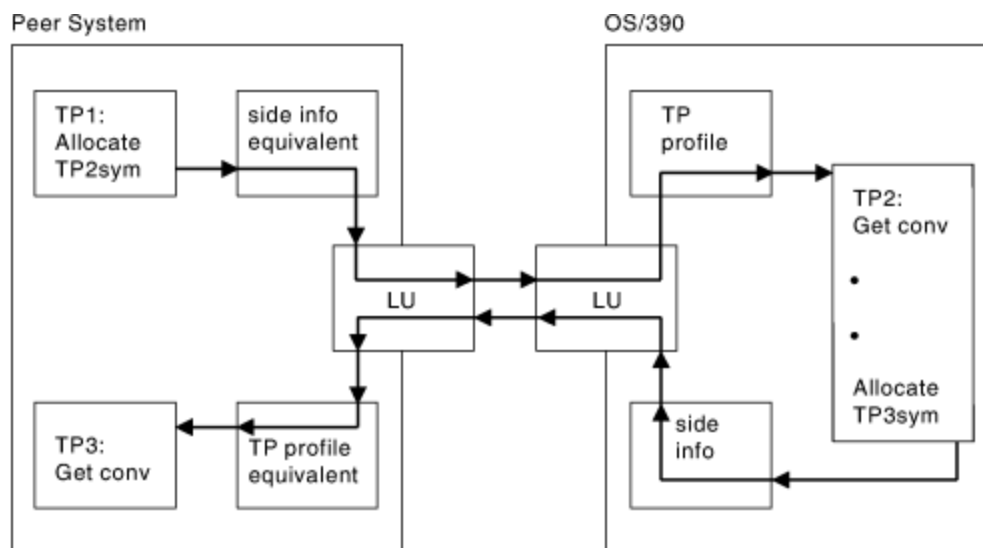


Figure 9. Using TP Profiles and Side Information to Find a Partner TP

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.

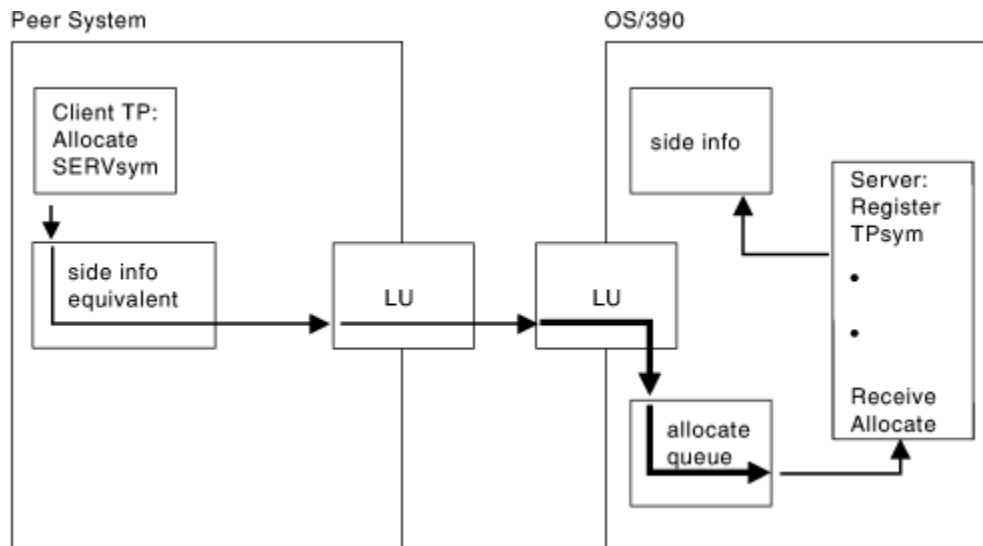


Figure 10. Using Side Information in Client/Server Communications

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 [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.

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 14 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.

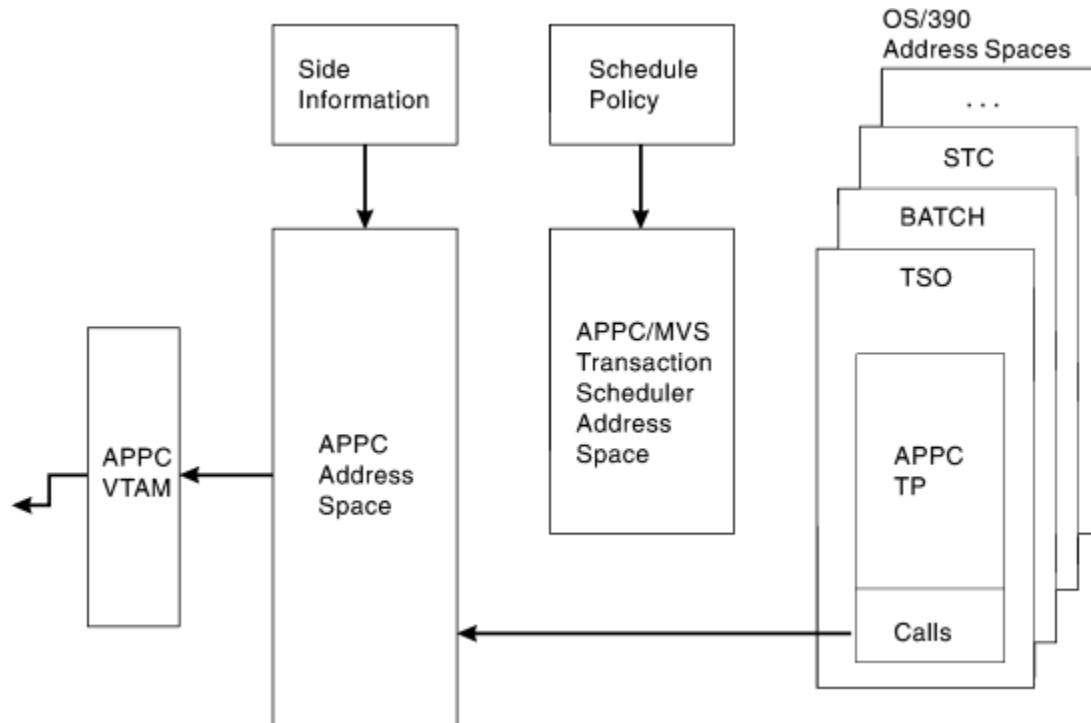


Figure 11. APPC/MVS Communications Services (Outbound)

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.

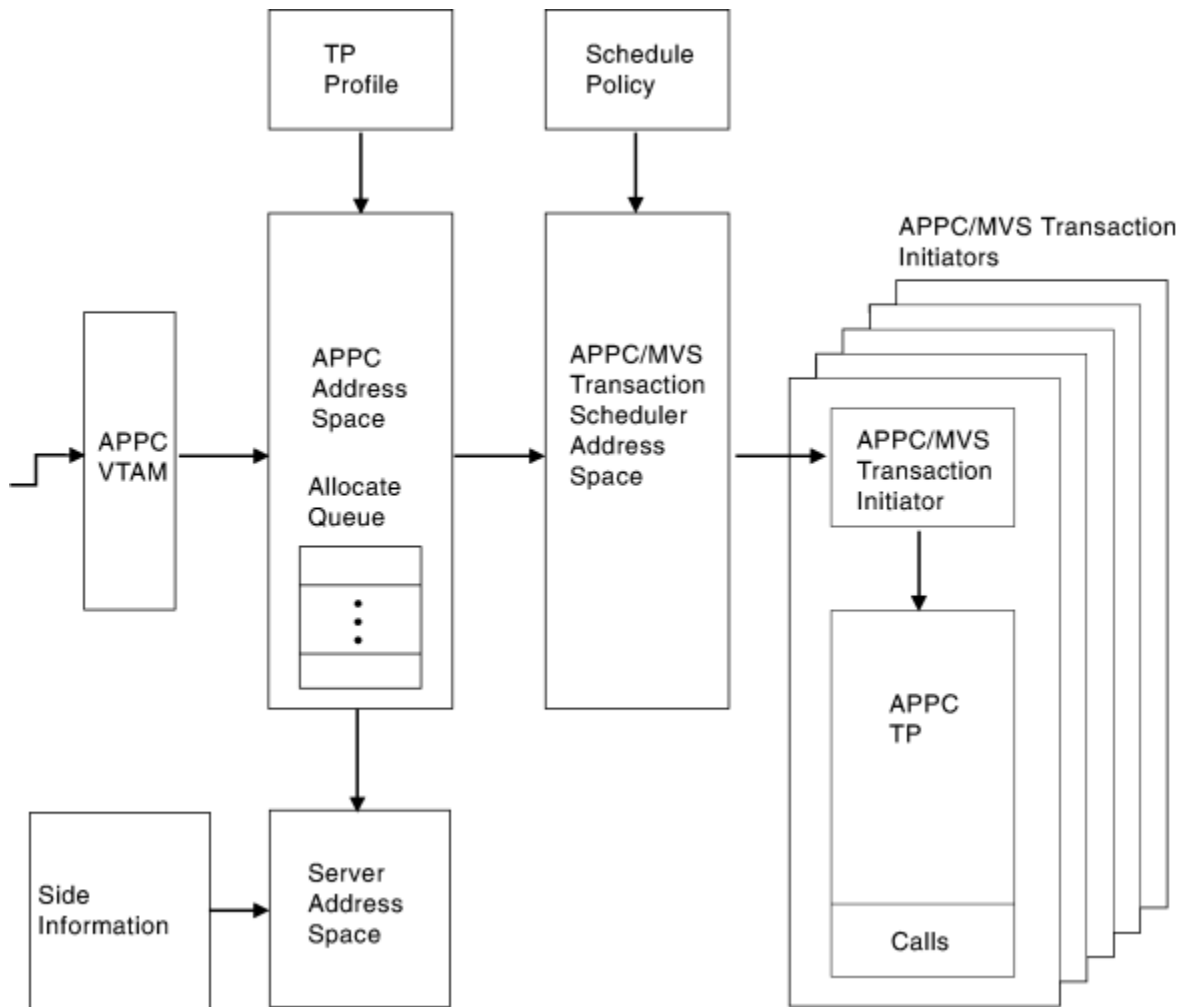


Figure 12. APPC/MVS Communication Services (Inbound)

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 15](#) shows these initiators on the right-hand side.

Steps to Write and Install an APPC/MVS Transaction Program

The following is an overview of the main steps to follow when designing, writing, and testing transaction programs for use with APPC/MVS. Later chapters of this book give the details about these steps. For information about writing APPC/MVS servers, see [z/OS MVS Programming: Writing Servers for APPC/MVS](#).

Application Programming Steps

1. Make basic design decisions:

- What functions (including MVS services or data) do you want the application to provide?
- Do you want to use CPI Communications for portability, use APPC/MVS services for MVS-specific functions like the use of data spaces or asynchronous processing, or combine the two types of services?
- Will the transaction program on z/OS run under the APPC/MVS transaction scheduler with a schedule type of standard or multi-trans, will it run under another transaction scheduler, or will it be processed by an APPC/MVS server?

2. Code the transaction program and its partner:

- Code the APPC/MVS TP to hold a conversation with a partner program, using appropriate callable services based on your design decisions.
- Code the partner program and have it installed on the desired system. Ensure that the appropriate system programming steps are taken (Steps 1-3 shown in [“System Programming Steps” on page 16](#)) if the partner program is to run under the APPC/MVS transaction scheduler.
- Test the transaction program and its partner:
 - For inbound TPs, write a TP test shell or use TSO/E TEST.
 - Optionally, supply TP profile information.
 - Supply side information, if the TP allocates a conversation using a symbolic destination name.
 - Test an inbound TP under its test shell or under the control of a user-level TP profile.

System Programming Steps

For details about these system programming steps needed to prepare your MVS/ESA system for APPC/MVS communications, see [z/OS MVS Planning: APPC/MVS Management](#).

1. Create one or more TP profile files and make entries for all inbound APPC/MVS transaction programs that are to be scheduled in response to inbound allocate requests. Inbound requests that are destined for an APPC/MVS server are not scheduled, and therefore do not require a TP profile.

2. Create a side information data set and make entries for any symbolic destination names that local programs use to identify their partners on outbound allocate requests or Register_For_Allocates requests.
3. Define local LUs and associate a TP profile file name and scheduler for them through LUADD statements in APPCPMxx members of the parmlib concatenation. You can also define LUs that are not to be associated with a transaction scheduler (with the NOSCHED option on the LUADD statement).
4. Define the APPC/MVS local LUs in SYS1.VTAMLST and the logon mode names in SYS1.VTAMLIB.
5. Define classes for the APPC/MVS transaction scheduler in parmlib member ASCHPMxx, and assign TPs to those classes in their TP profile entries.
6. Optionally, define LUs, TPs, and APPC/MVS servers to RACF for security checking. For information about defining security for APPC/MVS servers, see [*z/OS MVS Programming: Writing Servers for APPC/MVS*](#).

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.

Part 2. Programming

Chapter 2. Designing and Writing an APPC/MVS Transaction Program

APPC/MVS extends IBM's Advanced Program-to-Program Communications (APPC) support to the MVS/ESA operating system. With APPC/MVS, application programmers can include MVS programs in cooperative processing applications with partner programs running on the same z/OS system, other z/OS systems, or other operating systems.

This chapter covers the following topics for designing and writing APPC/MVS transaction programs:

- Benefits of APPC/MVS, including types of applications that lend themselves to APPC/MVS
- Conversation services that transaction programs use to communicate
- Flow diagrams of typical conversations
- Environment and design considerations, including:
 - Required processing environment
 - Portability and MVS-specific services
 - Transaction scheduling
 - Conversation security
 - Data formatting
 - Data conversion
 - Error handling and recovery.

Benefits of APPC/MVS for Application Programs

Many types of applications are good candidates for using APPC/MVS services. The most obvious candidates are applications that need to link MVS functions or data with processing done on other computers in an SNA network. When an MVS program must communicate with a program on another system, APPC/MVS can provide a temporary connection, freeing you from having to make a permanent connection between the programs or create a special access method.

A prime example of a cooperative processing application involving APPC/MVS is one in which a transaction program on an z/OS server provides z/OS services and data to a user interface program on a workstation. For example, a database-accessing program on MVS could receive requests from a partner program on the workstation, access an MVS database, perform intensive computations, and send results back to the partner program. This cooperative application would combine the usability of the workstation, with its end-user interface features such as graphics and windows, with the processing power and resources of the z/OS server.

The APPC/MVS part of an APPC cooperative application—an APPC/MVS transaction program (TP)—is any program in any address space on MVS that issues APPC/MVS or CPI Communications calls. The TP can run in task or SRB mode and can use other MVS services. A TP can also:

- Be scheduled in response to inbound allocate requests
- Initiate multiple, concurrent, and asynchronous APPC service requests
- Use JES SYSOUT and Job Submit facilities (only inbound TPs using the APPC/MVS scheduler)
- Use certain TSO/E programming services and TSO/E command facilities.

An APPC/MVS application can be a powerful adjunct to existing subsystems and environments such as TSO/E, IMS, CICS and DB2. An APPC/MVS application can create or access the following types of data:

- New or existing VSAM, sequential or partitioned data sets can be created, read, or updated through dynamic allocation or the TP profile.

- To update and extract CICS VSAM data, a conversation to a CICS transaction that uses CICS file control may be required. In some cases it may be possible to read CICS VSAM data using dynamic allocation, assuming a data disposition of SHR and the possibility of updates in progress.
- IMS data may be accessed by a conversation request to an IMS application, through an IMS scheduler, to either extract or update DL/I data. Direct access is possible if the APPC application is coded to the rules for an IMS BMP.
- DB2 data may be accessed directly through the call attach facilities provided by DB2.

APPC/MVS TPs may also participate in resource recovery by using the CPI Communications or MVS Commit or Backout services to synchronize changes with partner TPs.

In addition, APPC/MVS applications can use MVS facilities such as shared data spaces, hiperspaces or look-aside techniques to enhance performance and sharing of data across APPC/MVS transactions.

With this capacity for joining high-end data processing with end-user interfaces, APPC/MVS lends itself to compute-intensive and I/O- and data-intensive programs such as:

- Planning and control programs
- Knowledge-based processing
- Collector applications (example: gathering regional sales data)
- Distributor applications (example: price dissemination)
- Exception/alert processing
- Large sorts
- Monitoring/report generation
- Graphics
- Decision support (spread-sheet) applications
- Batch application scheduling
- Extensions of current subsystem environments

In addition to making cross-system connections, APPC/MVS presents a consistent interface for communications within the same z/OS system. A local TP can use APPC/MVS to hold a conversation with a partner TP that is under APPC/MVS control on the same system. The two TPs can be either:

- Defined to the same VTAM logical unit (LU=OWN)
- Defined to different VTAM logical units that are on the same z/OS system (LU=LOCAL).

A good analogy for this type of intra-MVS communication is a hotel telephone network; when a guest wants to call another guest in the same hotel, the hotel switchboard puts the call through to that room directly, and they have a conversation without using the public phone system at all.

Before designing a new application or adapting an existing one to take advantage of APPC/MVS, it is important to understand how the parts of the application communicate; what the partners can exchange in a conversation, and how they do it.

The Elements of Conversation

The TPs that make up an APPC cooperative processing application communicate through a conversation that takes place across a standard interface, just as people communicate through a conversation over the telephone. One partner calls the other, identifying it in a way that the system recognizes; the system makes the connection and the partner accepts the conversation. The conversation follows a protocol, with statements dictated by sequence and the state of the conversation. For example, one partner talks (sends data) while the other listens (receives data). They take turns sending and receiving data until one of them ends the conversation.

APPC conversations follow a **half-duplex** protocol. That means data can be transmitted back and forth between partners, but only in one direction at a time. To make sure that partners do not get out of sequence or both try to communicate at the same time, APPC enforces conversation states.

Conversation States

The state of a conversation depends on what a transaction program or its partner has just done and determines what actions a TP can take next.

The basic conversation states in APPC are:

State

Description

Reset

The initial state, before communications begin or after they end.

Send

The state in which a program is allowed to send data.

Receive

The state in which a program is allowed to receive data.

Confirm

The state in which a program must respond to its partner.

Some of the callable services cause a transition from one state to another. For example, when a local program in Reset state issues a successful Allocate call, the local program goes into Send state. The partner goes into Receive state when it issues CMACCP or the Get_Conversation service to accept the conversation.

APPC/MVS implements these states slightly differently for its CPI Communications and MVS TP conversation calls. For detailed descriptions of the conversation states that apply to all the individual conversation calls, see the following:

- For CPI Communications, the section about CPI communications terms and concepts, and the state table in *CPI-C Reference*
- For MVS TP conversation calls, [Chapter 4, “The APPC/MVS Programming Interface,”](#) on page 45 and [Appendix C, “APPC/MVS Conversation State Table,”](#) on page 399.

APPC Conversation Services

This section describes the basic APPC services that TPs use in a conversation.

Starting a Conversation

To start (allocate) a conversation with its partner, a TP calls the Allocate service, identifying its partner and requesting that a connection be made. This is analogous to dialing a telephone. The caller can use a symbolic destination name, similar to a phone number, to identify the partner; the caller does not need to know where the partner is physically located.

APPC makes the connection if possible and notifies the partner of the conversation request. The partner can find out who is requesting the conversation through the Get_Conversation service, and can either accept the conversation (analogous to picking up the phone and talking) or reject it (analogous to hanging up after finding out who's calling). Once the conversation is accepted, either TP can use the Get_Attributes service to obtain additional information about its current partner and conversation.

Sending and Receiving Data

If the partner TP accepts the conversation, the caller goes into Send state and the partner goes into Receive state. The caller can then use the Send_Data service, putting data into buffers and requesting it be sent. APPC/MVS sends the data and notifies the receiving partner when the data arrives.

The receiver can use the Receive_and_Wait service to wait for inbound data to arrive, or use the Receive_Immediate service to receive any data that is available without waiting.

The format of the data depends on the type of conversation. In a *basic conversation*, the caller formats the data into separate records, specifying the length and data of each record. In a *mapped conversation*,

the caller simply provides the data and lets APPC format it. The sender and receiver need to agree in advance on which type of conversation they will use.

When a TP uses the Send_Data service, APPC tries to optimize data transmission by waiting until buffers are full or until the TP performs some other conversation service. The TP can also use the Flush service to remain in send state while forcing APPC to send buffered data immediately.

Requesting Permission to Send

A local TP in Receive state can request permission to send data, for instance, in response to something it received. APPC informs the partner of the request, which the partner can grant or ignore. To grant the request, the partner issues a Receive service, thus entering Receive state and putting its partner into Send state.

Granting Permission to Send

A TP in Send state can use the Prepare_to_Receive service to put itself in Receive state and put its partner in Send state. A TP can achieve the same result by using the Receive_and_Wait service, but Receive_and_Wait requires a receive buffer as input, while Prepare_to_Receive does not.

Requesting Confirmation

TPs can synchronize their communications by requesting and granting confirmations. In some situations, a TP might need to confirm that its partner has received and successfully processed data already sent. For example, a TP that is uploading data from a workstation to the host would not want to delete the data until it is sure its partner on the host received it. When the sending TP requests a confirmation, APPC sends any buffered data to the partner. When the partner receives all the data, the partner also receives the confirmation request; the partner can then grant the confirmation or reject it by sending an error notification. If a conversation is to include confirmations, the requesting TP must indicate that at the start of the conversation.

Sending Error Notification

When a TP encounters an error or cannot grant a confirmation, the TP can send an error notification to its partner, from either Send or Receive state. For example, if TP A is sending records to TP B and then deleting them, TP B should notify A if an internal error prevents B from filing a record. Otherwise, TP A might delete the record and lose it forever. To notify partners and recover from such errors, TPs use the Send_Error service.

Returning Error Information

When APPC/MVS returns a return code that indicates an error in a call to a conversation callable service, your application can call the Error_Extract service to return detailed information about the error. For information about how to use Error_Extract see [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77](#).

Ending Conversations

When a TP is finished communicating with its partner, it should end the conversation with the Deallocate service, which is analogous to hanging up in a phone conversation. The partner then receives a deallocation indicator and any remaining data. The partner goes into Reset state and typically finishes its own processing.

Identifying TP Partners to MVS

After TPs are written, they need to be defined to the system before they can communicate. The system needs to know where partner TPs are located and needs other characteristics to schedule the TPs properly. That information goes in TP profiles and side information files on MVS.

Supplying TP Profiles on MVS

If a TP on MVS is the target of an allocate request from its partner, the target (inbound) TP needs a TP profile containing routing and scheduling information. APPC/MVS uses the TP profile to locate and initialize the inbound TP.

Supplying Side Information on MVS

TPs can specify symbolic names for their partners when allocating a conversation. The symbolic name must correspond to an entry in a side information data set that contains the partner's name, location and logon mode. The symbolic name frees the TP programmer from having to know that information.

For more details about TP profiles and side information, see [z/OS MVS Planning: APPC/MVS Management](#).

Relating MVS Callable Services to CPI Communications

Table 1 on page 25 shows how APPC callable services on MVS relate to CPI Communications. The MVS TP services (ATBxxxx) are MVS-specific and make use of the MVS architecture. Calls to CPI Communications (CMxxxx) are the same on many systems and TPs can use them for portability.

<i>Table 1. Mapping of MVS TP Services and CPI Communications</i>	
MVS TP Conversation Services	CPI Communications
Allocate	CMINIT (Initialize_Conv) together with CMALLC (Allocate)
Confirm	CMCFM
Confirmed	CMCFMD
Deallocate	CMDEAL
Error_Extract	(no CPI equivalent)
Flush	CMFLUS
Get_Attributes	CMECS (Extract_Conversation_State), CMEMN (Extr_Mode_Name), CMEPLN (Extr_Part_LU_Name), CMESL (Extr_Sync_Level)
Get_Conversation	CMACCP
Get_TP_Properties	(no CPI equivalent)
Get_Type	CMECT
Post_on_Receipt	(no CPI equivalent)
Prepare_to_Receive	CMPTR
Receive_Immediate	CMRCV
Receive_and_Wait	CMRCV
Reject_Conversation	(no CPI equivalent)
Request_to_Send	CMRTS
Send_Data	CMSEND
Send_Error	CMSERR

Table 1. Mapping of MVS TP Services and CPI Communications (continued)	
MVS TP Conversation Services	CPI Communications
Set_Conversation_Accounting_Information	(no CPI equivalent)
Set_Syncpt_Options	(no CPI equivalent)

If you would also like to see how these and other APPC/MVS callable services are related to the SNA LU 6.2 architecture, refer to [Appendix D, “Support for SNA LU 6.2 Verbs and Option Sets,”](#) on page 419.

Flow Diagrams of Typical APPC/MVS Conversations

The following figures show the flow of control in some of the most typical kinds of conversations. These examples use the MVS TP conversation calls; for similar flows using CPI Communications, see [Figure 3](#) on page 5 and *CPI-C Reference*.

In these flows, the first occurrence of each callable service (ATBxxxx) is accompanied by the corresponding conversation service name, for example, the Allocate service. The callable services are abbreviated to show only the pertinent parameters. For clarity, the parameter values appear in parentheses after the parameter name; in actual syntax, the values are positional.

Simple One-Way Conversation

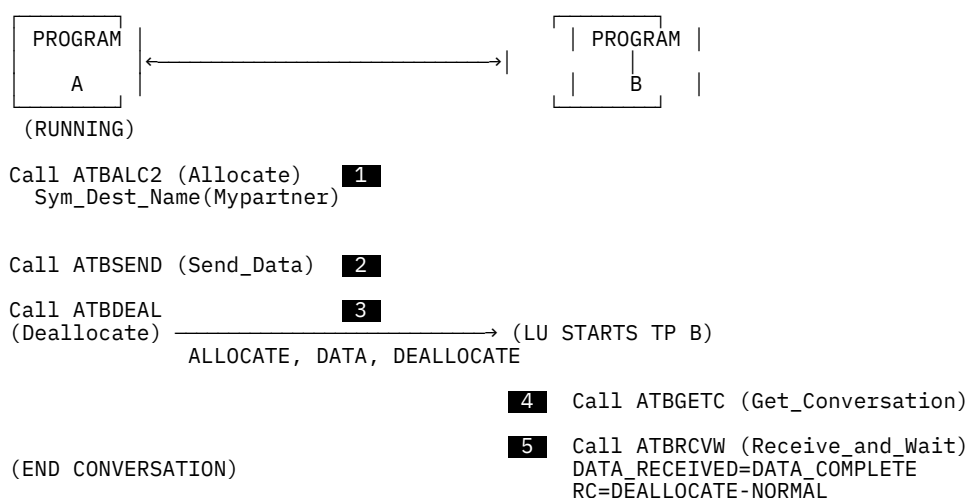


Figure 13. One-Way Conversation

Any APPC conversation includes three phases: initialization, data exchange, and termination. In [Figure 13](#) on page 26, a local MVS transaction program (program A) initiates the conversation, sends data to its partner (program B), and ends the conversation.

1. The Allocate service initializes the conversation, specifying the partner with a symbolic destination name. That name corresponds to a side information entry naming program B and identifying program B's LU name and the session mode name.
2. The Send_Data service specifies that a block of data be put in a buffer for sending.
3. The Deallocate service notifies APPC that this is the end of the conversation, forcing the LU to send the buffered data. The output from the three above services (allocate, data, deallocate) crosses the network; APPC uses the output from the Allocate service to initialize program B.
4. Program B issues the Get_Conversation service to get information about the caller (program A).
5. Program B issues the Receive_and_Wait service to receive the data. The data_received parameter and return code tell program B that the data was complete and that the conversation was deallocated.

Simple Two-Way Conversation

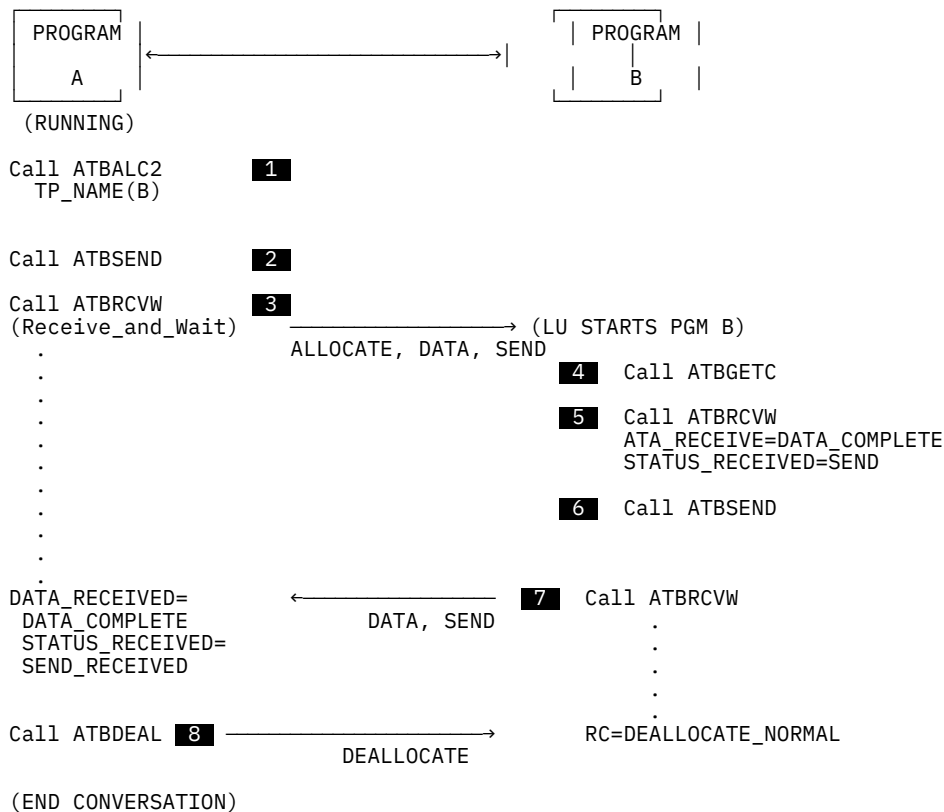


Figure 14. Two-Way Conversation

Figure 14 on page 27 shows a more complete conversation, in which both partners send and receive data.

1. Program A allocates a conversation with program B.
2. Program A sends data as in Figure 13 on page 26.
3. Program A calls `Receive_and_Wait` to wait for data from program B, changing its own state from `Send` to `Receive` and forcing its LU to send buffered data.
4. Program B calls the `Get_Conversation` service to accept the conversation.
5. When program B receives the data, its `status_received` parameter indicates that it has entered `Send` state (caused by A entering `Receive` state) and B can now send data.
6. Program B calls the `Send_Data` service, with data that the LU puts in a buffer.
7. Program B calls the `Receive_and_Wait` service, entering `Receive` state again, and forcing the LU to send the buffered data.
8. Program A receives the data, checking for completion. Program A then ensures that the call to `ATBRCVW` receives a `status_received` of `send_received`, which is necessary to deallocate the conversation (because Program A can call `Deallocate` only in `Send` state). Then program A deallocates the conversation. Program A enters `Reset` state, as does program B when its return code from `Receive_and_Wait` indicates the deallocate request was received.

Confirmation of a Transaction

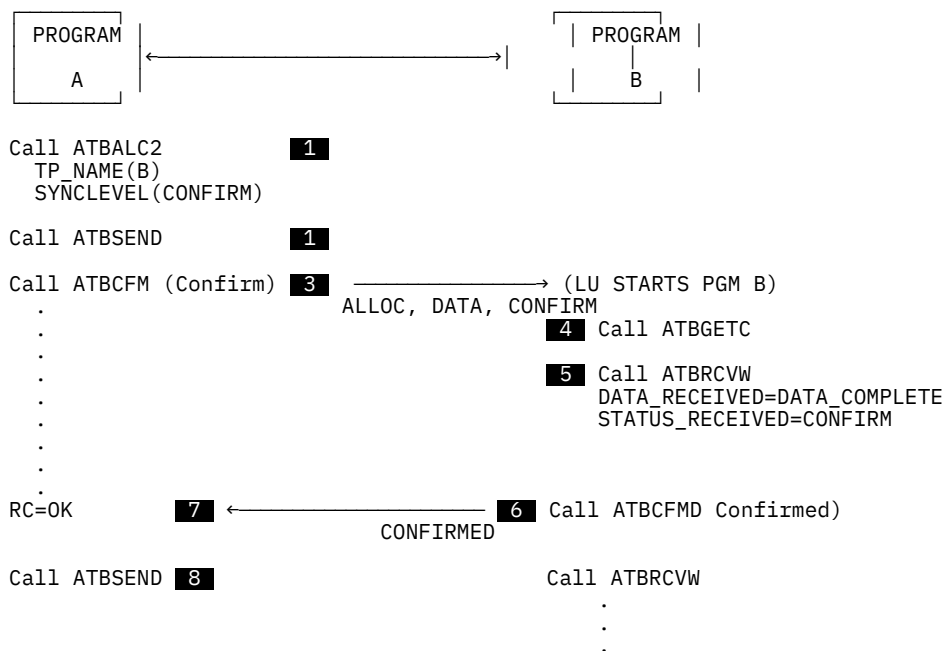


Figure 15. Example of a Confirmed Transaction

A transaction program can request that its partner confirm that all the data sent so far has been received and processed successfully. In [Figure 15 on page 28](#):

1. Program A allocates a conversation with program B, setting the Sync_level parameter to CONFIRM to allow confirmation processing on the conversation.
2. Program A sends data to program B.
3. Program A calls the Confirm service. The confirmation request forces the LU to send buffered data.
4. Program B calls the Get_Conversation service to accept the conversation.
5. Program B receives the data, checking for completion, and the status received. The status indicates that a confirmation has been requested.
6. Program B calls the Confirmed service, granting the confirmation. It could have sent an error notification instead, if something was wrong.
7. The confirmation results in a return code of OK for program A's confirmation request.
8. Program A continues sending data.

Sending Error Notification

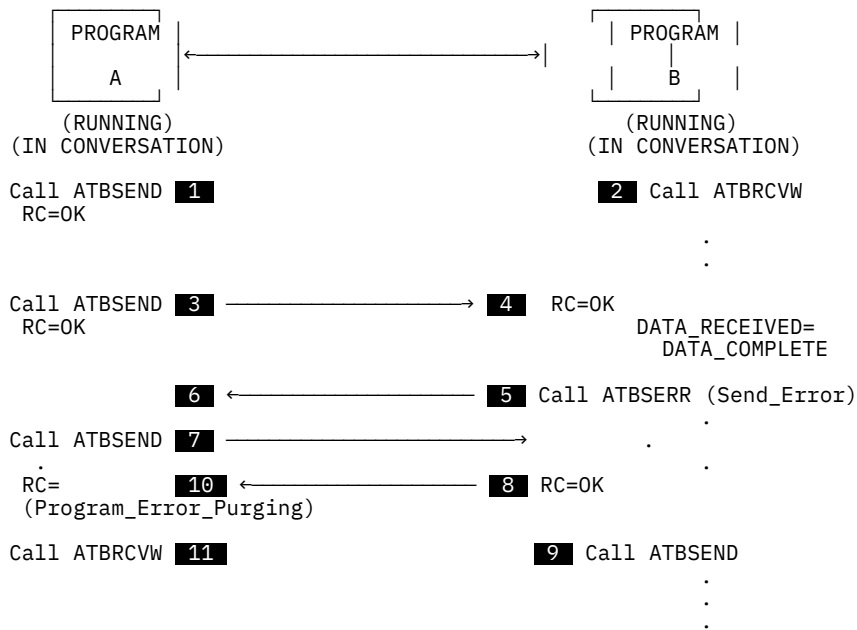


Figure 16. Example of Send_Error in Receive State

A transaction program can send an error notification to its partner to report that an error occurred and to cause buffer data to be purged.

In Figure 16 on page 29, program A has already allocated the conversation and is in Send state, with program B in Receive state.

1. Program A calls the Send_Data service, causing the LU to place the data (a logical record) in its buffer. Nothing is sent.
2. Program B calls the Receive_and_Wait service, suspending processing until it receives data.
3. Program A calls Send_Data again, causing the LU to place more data (another logical record) in its buffer. The LU now has enough data to send, based on session characteristics, so it sends the data.
4. The LU returns control to program B, indicating that the program has received a complete record.
5. Program B encounters an error in the data or in its processing and calls the Send_Error service. The Send_Error service causes program B's LU to purge information it has received but not yet sent, and to send a negative response. Program B's processing is suspended awaiting Send control.
6. Program A's LU receives the negative response, purging any remaining buffered data from program A.
7. Program A, unaware of the error yet, calls Send_data, which fails. The LU does not accept the data. Instead, the LU sends Send control to program B, suspending program A.
8. The LU for program B receives the Send control, sends the error notification, and returns control to program B.
9. Program B calls the Send_Data service, possibly to transfer more error-recovery information.
10. Program A's LU returns control to it, along with the error notification (return code program_error_purging).
11. Now in Receive state, program A calls Receive_and_Wait.

TP Environment and Design Considerations

Your choice of APPC/MVS callable services to use in a transaction program depends on the purpose of the application and design considerations such as portability, use of MVS-specific services, and transaction scheduling. Those factors and their implications are discussed in the following sections.

As you think about the design of cooperative applications that use APPC/MVS, consider the following questions:

- Which part or parts of the application should run under MVS?
- For the MVS part of the application, do you want the code to be portable to other systems? Do you want to include services that are unique to MVS?
- When inbound requests for the program arrive, will the APPC/MVS transaction scheduler or another transaction scheduler initiate the program?
- Assuming you use the APPC/MVS transaction scheduler, what schedule type (standard or multi-trans) should you design for? Considerations include:
 - How often the program will be requested
 - How long it will take the program to run
 - How much resource allocation the program will require
 - Whether you can provide security for multiple partners.

You will need to decide whether it would be more efficient for the transaction program to be initialized and ended for each conversation (standard scheduling), or to remain active between conversations and serve different partners in sequence (multi-trans scheduling).

- Should the application use *basic* or *mapped* conversations? In other words, should it create logical records in a special format for transmitting data, or let APPC do the formatting?

The answers to these questions are fundamental to the design of an APPC/MVS TP. The following sections will help you find the answers.

The General APPC/MVS Environment for Transaction Programs

Any MVS program that calls APPC/MVS services, or is attached by an APPC/MVS LU in response to an inbound allocate request, is considered to be an APPC/MVS transaction program. The following is a description of the general processing environment for all APPC/MVS TPs, including requirements they must meet and features they can use.

Requirements for TPs in Problem-Program State

The following requirements apply to TPs that are written to run in problem-program state:

- APPC/MVS services must be invoked in 31-bit addressing mode.
- All parameters of APPC/MVS services must be addressable by the caller and in the primary address space, except for the *buffer* parameter of the Send_Data, Extract_Information, Receive_and_Wait, and Receive_Immediate services, which may reside in another address space or data space.

If you are writing a TP to run in problem-program state, you can skip over [“General Requirements” on page 30](#) and continue reading at [“Features of APPC/MVS for All TPs” on page 31](#).

General Requirements

The following general requirements apply to APPC/MVS services invoked by any TP but include requirements (such as cross memory allowed and locks not allowed) that are only of concern to TPs running in supervisor state or PSW key 0-7.

Authorization:

Supervisor state or problem state, any PSW key

Dispatchable unit mode:

Task or SRB

Cross memory mode:

Any PASN, any HASN, any SASN

AMODE:

31-bit

ASC mode:

Primary or access register (AR)

Interrupt status:

Enabled for I/O and external interrupts

Locks:

No locks held

Control parameters:

All parameters must be addressable by the caller and in the primary address space, except for the *buffer* parameter of Send_Data, Extract_Information, Receive_and_Wait, and Receive_Immediate services, which may reside in another address space or data space.

Features of APPC/MVS for All TPs

The following features of the APPC/MVS environment are available to all APPC/MVS TPs.

- Multiple conversations within a program

APPC/MVS transaction programs can hold one or more outbound conversations at the same time. There is no limit to the number of outbound conversations APPC/MVS TPs may have other than a limit imposed by the number of available sessions.

Generally, a TP is limited to conversing with only one inbound TP at a time. However, you can use either of the following methods to allow an application to process multiple inbound conversations concurrently:

- Schedule multiple copies of the same TP to converse with different partner TPs. It is then possible for several outbound TPs to allocate conversations with different instances of the same inbound TP.
- Design the TP as an APPC/MVS server. Such programs can process multiple inbound conversations in the same address space. For more information, see [z/OS MVS Programming: Writing Servers for APPC/MVS](#).

- Shared conversations across program boundaries

APPC/MVS transaction programs can call other programs and attach other tasks, and the called programs and attached tasks can do the same. In addition, APPC/MVS allows programs to share a single conversation across multiple tasks or SRBs in an address space.

APPC/MVS considers the scope of a TP to be the home address space. Access to a conversation is limited to programs whose home address space is the same as the home address space of the TP that allocated or accepted the conversation. However, the TP may access the conversation while also executing code in another address space.

The management and integrity of conversations within an address space is the responsibility of the programs within, and a TP is considered active until control returns to the system or the address space is terminated.

- Simultaneous execution of authorized and unauthorized TPs

If your application plans to run authorized and unauthorized TPs simultaneously in the same address space, the application must ensure that the integrity of conversations is maintained within the address space. When an authorized TP allocates a conversation, APPC/MVS does not protect the conversation from unauthorized TPs that are running on other dispatchable units within the address space. Therefore, unauthorized TPs can call APPC services on the conversation that is being used by the authorized part of an application, and might receive sensitive data or send incorrect data to the partner TP. (An authorized TP is one that runs in either supervisor state or PSW key 0-7.)

- Concurrent APPC requests

APPC/MVS allows concurrent or multiple APPC requests to be issued and outstanding at the same time within a transaction program, if the requests are for different conversations. An APPC request from a particular conversation is rejected by APPC/MVS when there is an outstanding request from the same conversation. There is one exception: you can issue a Deallocate call with a deallocate type of

Deallocate_ABEND to end a conversation at any time, when you have an outstanding APPC call from another task on that conversation.

- Deferred attention interrupts

All APPC/MVS callable services are protected from attention interrupts. If a user presses the attention interrupt key to interrupt the processing of a transaction program, server, or transaction scheduler after it has called an APPC/MVS callable service, the system defers the interrupt until the callable service processing is complete.

- Support for SRB-mode callers

Transaction programs in SRB mode may invoke both MVS-specific callable services and CPI Communications calls. All APPC/MVS services available to a task-mode caller are available to an SRB mode caller.

- JES and TSO/E services

All inbound TPs that are scheduled by the APPC/MVS transaction scheduler can use the same JES2 and JES3 services and subsystem interface (SSI) calls that are intended for use by batch jobs on MVS. For a list of intended SSI calls, see *z/OS MVS Using the Subsystem Interface*. Inbound TPs can use the same JES SYSOUT and data set integrity checking services available to other MVS applications, with the exception that SYSOUT data sets allocated by inbound TPs are treated as spin data sets, and SYSOUT data is printed only when the TP ends or the SYSOUT data set is deallocated or closed and freed. If the installation changes the default subsystem for APPC work, then these JES services may be unavailable to inbound TPs.

Standard APPC/MVS TPs that are scheduled by the APPC/MVS transaction scheduler can access many TSO/E services from an MVS (non-TSO) environment through the TSO/E environment service. For more information about the TSO/E environment service, including lists of the TSO/E services that it supports, see *z/OS TSO/E Programming Services*. A standard TP profile can also provide TSO/E services to an inbound TP by running the TSO/E Terminal Monitor Program.

Portability and MVS-Specific Services

One of the first decisions affecting the design of an APPC/MVS transaction is portability. If you want your transaction program to be able to run on other operating systems besides z/OS, you can use CPI Communications to make it more easily portable to other systems including AIX, OS/400, OS/2, and VM. APPC/MVS supports Common Programming Interface (CPI) Communications routines as documented in *CPI-C Reference* with deviations listed in [Chapter 3, “Using CPI Communications,” on page 37](#) of this document.

If you want your transaction program to be able to use specific features of MVS and provide them to transaction programs on other systems, the MVS TP conversation services might be more appropriate. For example, using MVS TP conversation services, an MVS transaction program can send data to a partner program from an MVS data space, and can specify asynchronous processing.

Both CPI Communications and MVS TP services support high-level languages including COBOL, C, FORTRAN, PL/I, and REXX. You can combine calls to CPI Communications and MVS-specific services in the same conversation from a TP written in one of those languages. Calls to the MVS-specific services will not change any CPI Communications conversation characteristics previously established by a CPI Communications call. Any combination of MVS-specific services with CPI Communications calls will affect portability, because the MVS services will not work on other systems.

One way to minimize their affect on portability is to call MVS-specific services from subroutines that can be bypassed in non-MVS environments.

Features of the MVS-Specific Services

The following are features of APPC/MVS that TPs can get from the MVS-specific (ATBxxxx) services only:

- Support for data spaces

APPC/MVS allows send and receive data buffers to reside in data spaces and accommodate large amounts of data. The following LU 6.2 conversation services use data buffers and accept an access list entry token (ALET), along with the buffer address, to designate the address space or data space in which the buffer resides:

- Extract_Information
- Receive_and_Wait
- Receive_Immediate
- Send_Data

The services listed above accept ALETs for entries on the dispatchable unit access list (DU-AL) for the unit of work that calls those services. Keep the following restrictions in mind when specifying an ALET:

- The ALET cannot be the value 1 (which indicates *secondary ASID*).
- The ALET can represent an entry on the primary address access list (PASN-AL) only when the ALET points to a SCOPE=COMMON data space (also known as a common area data space).

A SCOPE=COMMON data space provides your TP with virtual storage that is addressable from all address spaces and all programs. Your TP might want to use a common area data space when it has large amounts of data that it wants to share across multiple address spaces. For more information about how to create and use SCOPE=COMMON data spaces, see [z/OS MVS Programming: Extended Addressability Guide](#).

- Parameter lists and parameters other than data buffers must reside in the caller's primary address space.
- Asynchronous Processing

TPs can use many of the APPC/MVS services asynchronously by specifying the address of an event control block (ECB) on the Notify_Type parameter. The TP can then continue other processing without waiting for the service to complete. When the service completes, APPC/MVS notifies the TP by posting the ECB.

- Transaction Schedule Types

The APPC/MVS transaction scheduler lets you assign a schedule type of standard or multi-trans to transaction programs.

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. Standard scheduling requires that a TP's resources be allocated and deallocated for each inbound conversation, and isolates TPs from each other and from subsequent requests for the same transaction program. The standard schedule type provides full security and basic performance for TPs that are not invoked very frequently or do not require extensive allocation of resources.

In contrast, the multi-trans schedule type causes a transaction program to remain active between inbound conversations, with its resources available. After each conversation ends, the next calling partner can use the same instance of the transaction program and avoid the overhead of repeated resource allocation and deallocation.

Multi-trans TPs run under a shell environment that is responsible for doing all necessary cleanup to ensure that a TP's conversations and data are secure for consecutive users. With proper design, multi-trans scheduling can offer greater performance for transactions that are requested often by multiple users, that have a high resource overhead, and that finish processing comparatively quickly.

To use the multi-trans scheduling option, transaction programs must be defined with a TP schedule type of multi-trans in the TP profile. For standard scheduling, the TP schedule type can be set to standard or omitted. For more information, see [“Using TP Schedule Types”](#) on page 59.

Note: Multi-trans scheduling is a feature of the APPC/MVS transaction scheduler. If your transaction program runs under another transaction scheduler on MVS, multi-trans scheduling is unavailable.

Security

APPC/MVS gives transaction programmers the ability to specify conversation security. On an Allocate call to a partner on MVS, you can specify the security environment under which the partner will run. Depending on the level of security you choose, the environment information may consist of a user ID, password and a security profile name. If RACF is installed on the system, the security profile name is treated as a RACF group name.

The installation can provide for security checking at the LUs, to verify inbound allocate calls and reject those that do not specify the required security environment information.

The APPC/MVS TP conversation services and CPI Communications provide different levels of conversation security on MVS. The APPC/MVS Allocate service allows a local TP to specify one of three possible levels of security when allocating a conversation, based on the partner program's security requirements. The level of security tells the partner LU what security environment information (if any) to verify and use.

- **Security_none**

Specifies that no user ID, password, or profile name is passed on the allocation request for the partner LU to verify or use.

- **Security_same**

Specifies that the partner TP should run under the same security environment (user ID and possible security profile name) as the local (calling) TP. The local TP's security environment may have been established at any of the following times:

- job initiation
- Start time
- LOGON time
- work context creation time
- explicitly requested by the caller through RACINIT
- if the caller is itself an allocated transaction program, at the time of its own allocation

- **Security_pgm**

Specifies that the local TP is providing a user ID, password, and security profile name to establish the security environment for the conversation. The partner LU can verify them.

Unlike the APPC/MVS Allocate service, the CPI Communications CMALLC call always allocates conversations using the security environment of the caller (*security_same*).

Using Basic or Mapped Conversations

Another design decision is whether to use a basic or mapped conversation. This decision affects the format and, potentially, the performance of data transmission.

Data travels between APPC transaction programs in buffers provided by the LUs involved. APPC defines a logical record format for the data as a sequence of length and data fields, with the 2-byte length fields (LLs) indicating the amount of data to follow before the next length field. The typical data pattern is “LL, data, LL, data.” The transaction programs can either format their data in this pattern themselves or leave the formatting to APPC, depending on the needs of the application.

Transaction programs that format their own data for transmission use what are called *basic conversations*. Those that let APPC do the formatting use *mapped conversations*.

- Basic conversations allow transaction programs to define logical records to their own specifications. The transaction programs can specify the exact lengths of their records using the “LL,data,LL,data” format and avoid the overhead of mapping done by APPC. Basic conversations give potential performance benefits and greater control.
- Mapped conversations are easier to code; they allow applications to send data in any format the partner programs expect. APPC itself maps the data into and out of the “LL,data,LL,data” logical record format for each transmission. Mapped conversations are recommended for most applications.

Both CPI Communications and LU 6.2 callable services support basic and mapped conversations. To specify a mapped or basic conversation, you can use the CMSCT (Set_Conversation_Type) service, or the Conversation_type parameter of the LU 6.2 Allocate service.

Data Conversion

When TPs communicate between a host system and a workstation, the data that they send must be converted between EBCDIC and ASCII. You can do the conversion on either end of the conversation, either before the data is sent or after it is received. One important design consideration is portability; if a TP is portable and may reside on a host or a PWS, its partner must know where the TP is located, to avoid doing unnecessary conversion in cases when both TPs are running in the same environment.

Using Protected Conversations

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, by meeting the requirements listed in *z/OS MVS Planning: APPC/MVS Management*.

To allow APPC/MVS TPs and their partner TPs to establish protected conversations:

- Refer to the overview of resource recovery in *z/OS MVS Programming: Callable Services for High-Level Languages*, which describes the protocol and the roles of resource recovery programs, which can include APPC/MVS TPs and their partners.
- Update existing, or code new, TPs to:
 - Issue Allocate calls with a Sync_level parameter value of 2 (syncpt), which identifies the conversation as protected. Both the local LU and partner LU must be enabled to support conversations with a synchronization level of syncpt, and the specified mode name, if any, must be a value other than SNASVCMG.
 - Issue Commit or Backout callable services to coordinate resource updates with partner TPs. See *z/OS MVS Programming: Callable Services for High-Level Languages* for the Commit and Backout service call names, syntax, and parameter descriptions; see [Appendix C, “APPC/MVS Conversation State Table,”](#) on page 399 for information about the conversation states in which a TP may call the Commit and Backout services.
 - Optionally issue the new Set_Syncpt_Options callable service to alter the default options that determine how APPC/MVS processes Commit and Backout requests. To determine the options in effect, your TP can issue the Get_TP_Properties (ATBGTP4) service.
- Evaluate existing code to determine whether logic changes are required to correctly handle new return codes from APPC/MVS or CPI-C services.

You might notice some performance impact for TPs that establish protected conversations and call the syncpoint services (Commit and Backout). The amount of impact depends on the frequency of syncpoint operations, the number of resource managers involved, and the scope of the distributed application (within one system, within a sysplex, across platforms). Generally speaking, the improved data integrity should outweigh the performance impact.

If your TP is a multi-trans TP, design it to issue a Commit or Backout service call before issuing the next call to the Get_Transaction or Return_Transaction service. Otherwise, APPC/MVS abnormally deallocates the TP's conversations and causes all protected resources updated since the last commit to be backed out.

Error Handling and Deallocation of Conversations

When a TP calls APPC/MVS services with incorrect parameters, unknown values, or in the wrong conversation state, APPC/MVS returns an error return code. In some situations, the description of that return code provides enough information to help you diagnose and correct the error. In those cases, you should design your TP so it handles error return codes and sends appropriate messages to end users.

Error_Extract Service

In situations where an error return code does not provide adequate problem determination information, your TP can call the Error_Extract conversation service to return detailed information about the error. Error_Extract provides reason codes and messages that describe errors in APPC/MVS conversations. If a partner system or TP sends error log information to MVS, Error_Extract can return the error log information and a product set identifier. For more information about calling the Error_Extract service, see [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77.](#)

Send_Error Service

Your TP can call the Send_Error service to notify a partner TP or system of errors. If a TP finds an error from which it cannot recover, it should have a recovery routine to deallocate the conversation. If a TP ends abnormally without deallocating the conversation, its partner may continue to send data or wait to receive data until APPC/MVS cleans up the address space of the ending TP. APPC/MVS deallocates any outstanding conversations in the address space with a deallocate type of ABEND_SVC at the following times:

- If any asynchronous verbs are outstanding at the end of a step, APPC/MVS deallocates the address space's conversations, and the job is flushed (that is, subsequent job steps are not processed).
- At end of a job. APPC/MVS cleans up the initiator address space and deallocates any outstanding conversations.

You should design each TP to be able to handle the unexpected deallocation of its conversation at any time. Also be sure to deallocate conversations as soon as they are no longer needed, to free sessions for use by other TPs in the LU. TPs should always deallocate conversations before ending.

API Trace Facility

Using the API trace facility, you can diagnose not only errors that occur during a specific call, but also problems with the conversation flow between the TP and its partners. Depending on the location of the partners, you might have to use the API trace facility together with tracing facilities provided for other platforms, such as OS/2.

API trace data includes entries for:

- Parameters and values specified on calls issued for APPC/MVS and CPI-C services, and values provided on return from those calls.
- The same diagnostic information that the APPC/MVS Error_Extract service provides for calls that return non-zero return codes.
- Parameters and values specified on START and STOP requests for the API trace facility.

Through ATBTRACE parameters, you can start tracing for only specific TPs or users, or for many TPs, many conversations, and many users.

For more information about the API trace facility, see [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77.](#)

Note: If you plan to use the programming interface for CPI Communications, read [Chapter 3, “Using CPI Communications,” on page 37](#) and [Chapter 5, “Installing and Testing Transaction Programs,” on page 67](#). If you *do not* plan to use CPI Communications, continue reading with [Chapter 4, “The APPC/MVS Programming Interface,” on page 45.](#)

Chapter 3. Using CPI Communications

Note

After reading this chapter, refer to the *Common Programming Interface Communications Reference* for complete details on coding CPI Communications routines.

Also read [Chapter 5, “Installing and Testing Transaction Programs,” on page 67](#) for information about supplying TP profiles and side information on MVS.

Like the MVS-specific TP services, CPI Communications lets your programs communicate with other programs on the same z/OS system, other z/OS systems, or other operating systems in an SNA network. However, CPI Communications has the advantage of portability: programs using CPI Communications can be moved between different systems with minimum changes.

CPI Communications in APPC/MVS

APPC/MVS supports the following CPI Communications calls:

- CMACCP (Accept_Conversation) accepts an incoming conversation.
- CMALLC (Allocate) allocates a conversation between a local program and a partner program, allocating a session between the local logical unit and the partner logical unit, if a session is needed.
- CMCFM (Confirm) sends a confirmation request to the partner program and waits for a reply, so the two programs can synchronize their processing.
- CMCFMD (Confirmed) sends a confirmation reply to the partner program, in response to a confirmation request.
- CMDEAL (Deallocate) deallocates the specified conversation. The session might or might not remain bound.
- CMECS (Extract_Conversation_State) extracts the conversation state for the specified conversation.
- CMECT (Extract_Conversation_Type) extracts the conversation type characteristic for the conversation and returns the value to the program.
- CMEMN (Extract_Mode_Name) extracts the mode name characteristic for the conversation and returns the value to the program.
- CMEPLN (Extract_Partner_LU_Name) extracts the partner LU name characteristic for the conversation and returns the value to the program.
- CMESL (Extract_Sync_Level) extracts the synchronization level characteristic for the conversation and returns the value to the program.
- CMFLUS (Flush) flushes the local LU's send buffer, forcing the LU to send any buffered data.
- CMINIT (Initialize_Conversation) initializes values for various conversation characteristics.
- CMPTR (Prepare_to_Receive) changes the program from Send to Receive state in preparation to receive data.
- CMRCV (Receive) receives data on the specified conversation.
- CMRTS (Request_to_Send) notifies the partner program that the local program is requesting to enter Send state for the conversation.
- CMSCT (Set_Conversation_Type) sets the conversation type characteristic for the conversation.
- CMSDT (Set_Deallocate_Type) sets the deallocate type characteristic for the conversation.
- CMSED (Set_Error_Direction) sets the error direction characteristic for the conversation.
- CMSEND (Send_Data) sends data on the specified conversation.
- CMSERR (Send_Error) informs the partner program that the local program has detected an error.

- CMSF (Set_Fill) sets the fill characteristic for the conversation.
- CMSLD (Set_Log_Data) sets the log data characteristic for the conversation.
- CMSMN (Set_Mode_Name) sets the mode name characteristic for the conversation.
- CMSPLN (Set_Partner_LU_Name) sets the partner LU name characteristic for the conversation.
- CMSPTR (Set_Prepare_to_Receive_Type) sets the prepare to receive type characteristic for the conversation.
- CMSRC (Set_Return_Control) sets the return control characteristic for the conversation.
- CMSRT (Set_Receive_Type) sets the receive type characteristic for the conversation.
- CMSSL (Set_Sync_Level) sets the synchronization level characteristic for the conversation.
- CMSST (Set_Send_Type) sets the send type characteristic for the conversation.
- CMSTPN (Set_TP_Name) sets the TP name characteristic for the conversation.

For guide and reference information about using CPI Communications routines, see *CPI-C Reference*. APPC/MVS implements CPI Communications as documented in that document, with the following exceptions:

- Data logging: calls to CMSLD (Set_Log_Data) and parameters dealing with the error log are accepted, but ignored.
- The CMTRTS call returns a `product_specific_error` when issued for APPC/MVS conversations that cross a VTAM network.
- APPC/MVS does not support conversations with a `sync_level` characteristic of `CM_SYNC_POINT`.
- The character set restriction on LU names, VTAM mode names, and TP names differs slightly from those prescribed in the *CPI-C Reference*.

LU names and mode names can contain uppercase alphabetic, numeric and special characters (\$, @, #), and must begin with an alphabetic or special character. IBM recommends that special characters be avoided because they display differently depending on the language code page in use.

TP names may contain any character from the 00640 character set except a blank. (Blanks can still be used as a trailing pad character, but are not considered part of the string). IBM recommends that the asterisk (*) be avoided in TP names because it causes a list request when entered on panels of the APPC administration dialog.

See [Appendix A, “Character Sets,” on page 387](#) for a table showing the allowable characters.

Invocation Details for CPI Communications

Calls to CPI Communications are supported from programs using the following high-level languages on MVS:

- C
- COBOL
- CSP (Application Generator)
- FORTRAN
- PL/I
- REXX
- RPG

Interface Definition Files (IDFs) for CPI-C Calls

IDFs (also called pseudonym files or headers) for the CPI Communications calls are provided as described below. The IDFs define calls, parameters, and variable values.

As shown in [Table 2 on page 39](#), the CPI-C IDFs provided by APPC/MVS are not named according to the CPI-C file names, which are described in *CPI-C Reference*. IBM recommends that you rename IDFs to the CPI-C name when you place the IDF in a high-level language macro library.

APPC/MVS provides the following IDFs:

<i>Table 2. IDFs Provided by APPC/MVS for CPI-C Calls</i>		
Language	In Member	CPI-C file name
C	ATBCMC in SYS1.SIEAHDR.H	CMC
COBOL	ATBCMCOB in SYS1.SIEAHDR.H	CMCOBOL
FORTRAN	ATBCMFOR in SYS1.SIEAHDR.H	CMFORTRN
PL/I	ATBCMPLI in SYS1.SIEAHDR.H	CMPLI
REXX	ATBCMREX in SYS1.SIEAHDR.H	CMREXX
RPG	ATBCMRPG in SYS1.SIEAHDR.H	CMRPG

The ATBCMC IDF is also shipped in the z/OS UNIX System Services HFS directory /usr/include.

The CSP interface definition file CMCSPP COPY is shipped by Cross System Product, Release 3.3.0 and above.

Transaction Program (TP) Environment

Any MVS program that calls APPC services (including CPI Communications routines), or is attached by an APPC/MVS LU in response to an inbound allocate request, is considered to be an APPC/MVS transaction program. To call CPI Communications routines, a transaction program must meet the following requirements.

Requirements for TPs in Problem-Program State

The following requirements apply to TPs that are written to run in problem-program state:

- APPC/MVS services must be invoked in 31-bit addressing mode
- All parameters of APPC/MVS services must be addressable by the caller and in the primary address space.

If you are writing a TP to run in problem program state, you can skip over [“General Requirements” on page 39](#) and continue reading at [“Calling CPI Communications Routines” on page 40](#).

General Requirements

The following general requirements apply to CPI Communications calls invoked by any TP but include requirements (such as cross memory allowed and locks not allowed) that are only of concern to TPs running in supervisor state or PSW key 0-7.

Authorization:

Supervisor state or problem state, any PSW key

Dispatchable unit mode:

Task or SRB mode

Cross memory mode:

Any PASN, any HASN, any SASN

AMODE:

31-bit

ASC mode:

Primary or access register (AR)

Interrupt status:

Enabled for I/O and external interrupts

Locks:

No locks held

Control parameters:

All parameters must be addressable by the caller and in the primary address space.

Transaction programs that call CPI Communications services while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section about providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

High-Level Language Compilers

Table 3 on page 40 shows a partial list of high-level language compilers that support CPI Communications calls on MVS/ESA. Calls can be made with other compiler levels and other compiler products that meet the preceding requirements and the linkage conventions described in [“Calling CPI Communications Routines”](#) on page 40. Note that the requirement for 31-bit addressing may limit some language functions that you can use.

Table 3. Some High-Level Language Compilers for CPI Communications Calls on z/OS	
Language	Compiler
C	C/370 Compiler Version 1, Release 2.0
COBOL	COBOL for OS/390 & VM Version 2
FORTRAN	VS FORTRAN Compiler Version 2, Release 6 0
PL/I	PL/I for MVS & VM Version 1, Release 1
RPG	RPG/370 Version 1, Release 1.0

Calling CPI Communications Routines

All CPI Communications calls have a general calling format as follows:

```
CALL routine_name      (parameters,return_code)
```

Some specific calling formats for languages that can invoke the services are:

COBOL

CALL “routine_name” USING parm1,parm2,...return_code

FORTRAN

CALL routine_name (parm1,parm2,...return_code)

C

routine_name (parm1,parm2,...return_code)

PL/I

CALL routine_name (parm1,parm2,...return_code)

REXX

ADDRESS CPICOMM “routine_name parm1 parm2 ...return_code”

For REXX, enclose the routine name and all parameters within one pair of single or double quotes. Parameters must be initialized to appropriate values. The host command environment resolves the parameter values. For more information, see [z/OS TSO/E REXX Reference](#).

RPG

For RPG, begin the CPI communications call in column 28, and enter the routine name in columns 33-42. For each parameter, list the PARM keyword in column 28 and the parameter value in columns 43-48, as follows:


```

C          *OS      CALL "routine_name"
C          PARM      parm1
C          PARM      parm2...
C          PARM      return_code

```

Any high-level language that conforms to the following linkage conventions may be used to issue CPI Communications calls on MVS:

- Register 1 must contain the address of a parameter list, which is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.
- Register 13 must contain the address of an 18-word save area.
- Register 14 must contain the return address.
- Register 15 must contain the entry-point address of the service being called.
- If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.

On return from the service, general and access registers 2 through 14 are restored (registers 0, 1 and 15 are not restored).

Parameter Descriptions

All of the parameters of the callable services are required, positional parameters. Some programming languages let you pass literal values in these parameters. None of the parameters are optional.

APPC/MVS checks all parameters for valid values, regardless of whether the parameters are used in call processing. For example, the locks parameter on the Prepare_to_Receive service must contain either a 0 or 1 (the only valid values), otherwise APPC/MVS rejects the request to change to receive state with a program_parameter_check return code, regardless of what value is supplied in Prepare_to_receive_type.

See *CPI-C Reference* for descriptions of the parameters of the CPI Communications services.

Required Modules

Transaction programs using CPI Communications on MVS, other than those written in REXX, must either:

- Be link-edited with the load module ATBPBI, which is provided in SYS1.CSSLIB, or
- Issue the MVS LOAD macro for the APPC/MVS service to obtain its entry point address, then use that address to call the APPC/MVS service.

Conversation States

The conversation states for CPI Communications differ slightly from those used for MVS-specific TP services. For complete descriptions of the conversation states that apply to the CPI Communications calls, see *CPI-C Reference* Chapter 2 and Appendix C.

Performance Considerations

The relative performance speed of APPC/MVS callable services varies depending on the functions that the callable service performs. For example, services that call VTAM or cause the movement of data buffers involve a greater number of internal instructions. For an overview of performance considerations for calls to CPI Communications, see [Table 4 on page 42](#). For a similar chart of performance considerations for the MVS TP services, see [Table 5 on page 56](#).

Table 4. Performance Considerations for APPC/MVS CPI Communications Calls

CPI Call	Calls VTAM	Causes DASD I/O	Causes buffer moves	Calls RACF	Creates SMF record
CMINIT	No	Sometimes(1)	No	Sometimes(3)	No
CMALLC	Yes	No	No	RACROUTE= TOKENXTR, TOKENMAP	No
CMCFM	Yes	No	No	No	No
CMCFMD	Yes	No	No	No	No
CMDEAL	Yes	No	No	No	No
CMFLUS	Yes	No	No	No	No
CMACCP	No	No	No	No	No
CMECS	No	No	No	No	No
CMECT	No	No	No	No	No
CMEMN	No	No	No	No	No
CMEPLN	No	No	No	No	No
CMESL	No	No	No	No	No
CMPTR	Yes	No	No	No	No
CMRCV	Yes	No	Yes	No	No
CMRTS	Yes	No	No	No	No
CMSEND	Yes	No	Yes	No	No
CMSERR	Yes	No	No	No	No
CMSCCT	No	No	No	No	No
CMSxxx(2)	No	No	No	No	No

Table 4. Performance Considerations for APPC/MVS CPI Communications Calls (continued)

CPI Call	Calls VTAM	Causes DASD I/O	Causes buffer moves	Calls RACF	Creates SMF record
Note: <ol style="list-style-type: none"> 1. Might read from the side information file. (Entries are cached for quick retrieval after first reference). 2. All other CPI Communications Set_xxx calls. 3. Calls RACF only to verify access to side information file. 					

Chapter 4. The APPC/MVS Programming Interface

Like CPI Communications, APPC/MVS's MVS-specific services let your programs communicate with other programs on the same z/OS system, OS/390 systems, or other operating systems in an SNA network. Unlike CPI Communications, programs using the MVS-specific services are not portable to other systems. MVS-specific services make use of the MVS/ESA architecture, including data spaces and asynchronous processing, and also provide TP scheduling options, server functions, and test services.

The MVS-specific services are divided into the following categories:

TP Conversation Services

Provide access to all the conversation functions described in “The Elements of Conversation” on page 22. This set of services is to be used by transaction programs and has equivalent SNA LU 6.2 services and CPI Communications routines.

Advanced TP Services

Provide access to more advanced transaction program interfaces, including scheduling options and test services.

Allocate Queue Services

Provide access to LU 6.2 server functions, which allow you to direct inbound allocate requests to server address spaces. These services have no equivalent SNA LU 6.2 services or CPI Communications routines. The APPC/MVS allocate queue services are described in *z/OS MVS Programming: Writing Servers for APPC/MVS*.

System Services

Provide access to system services not normally used by transaction programs, but used by other MVS components, management subsystems, and transaction schedulers. The APPC/MVS system services are described in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

APPC/MVS TP Conversation Services

The APPC/MVS TP conversation services implement the APPC conversation functions in an MVS-specific way. The APPC/MVS TP conversation services correspond to the SNA models (verbs) described in the *SNA Transaction Programmer's Reference Document for LU 6.2*. The MVS TP conversation services let you combine many of the APPC conversation functions (Send, Flush, and so on) on a single call. They also support special features of the MVS architecture, such as allowing data buffers to reside in MVS data spaces, and providing asynchronous notification by event control block (ECB) when a service completes.

APPC/MVS TP Conversation States

The following is a list of the conversation states that apply to the APPC/MVS TP conversation services:

State

Description

Reset

The initial state, before communications begin.

Send

The program is able to send data or request a confirmation on this conversation.

Receive

The program is able to receive data on this conversation.

Send-Pending

The program has received both data and send capability on the same call.

Confirm

A confirmation request has been received on this conversation; that is, the partner program issued a Confirm call and is waiting for the local program to issue Confirmed. After responding with a confirmation, the local program enters **Receive** state.

Confirm-Send

A confirmation request and permission-to-send have both been received on this conversation; for example, the partner program issued a Prepare_To_Receive call with the prepare_to_receive_type set to *prep_to_receive_sync_level* and the sync_level for this conversation is *confirm*. After responding with a confirmation, the local program enters **Send** state.

Confirm-Deallocate

A confirmation request and deallocation notification have both been received on this conversation; that is, the partner program issued a Deallocate call with the deallocate_type set to *deallocate_sync_level* and the sync_level for this conversation is *confirm*. After responding with a confirmation, the conversation is deallocated.

Defer-Receive

The program enters **Receive** state after a synchronization call completes successfully. The synchronization call may be a Commit, Flush, or Confirm call.

Defer-Deallocate

The program has requested that the conversation be deallocated after a syncpoint operation has completed. The conversation is not deallocated until a successful syncpoint operation takes place.

Sync-Point

The program issued a Receive call and was given a return_code of OK and a status_received of TAKE_COMMIT. After a successful Commit operation, the program enters **Receive** state.

Sync-Point-Send

The program issued a Receive call and was given a return_code of OK and a status_received of TAKE_COMMIT_SEND. After a successful Commit operation, the program enters **Send** state.

Sync-Point-Deallocate

The program issued a Receive call and was given a return_code of OK and a status_received of TAKE_COMMIT_DEALLOCATE. After a successful Commit operation, the conversation is deallocated, and the program enters **Reset** state.

These conversation states apply to the individual MVS TP conversation services as shown in [Appendix C, "APPC/MVS Conversation State Table,"](#) on page 399.

Guide to the Conversation Services

APPC/MVS provides the following conversation callable services that TPs use to communicate in conversations. These services apply to both mapped and basic conversations, as specified on the Conversation_type parameter of the Allocate service.

Allocate

Allocate (start) a conversation with a partner TP.

Confirm

Request a confirmation, for example, that the partner has successfully received data that was sent.

Confirmed

Send a confirmation, in response to a request for confirmation.

Deallocate

Deallocate (end) a conversation.

Error_Extract

Return detailed information about errors indicated by APPC/MVS return codes.

Flush

Flush the LU's data buffer, forcing buffered data to be sent.

Get_Attributes

Get attributes of a specified conversation.

Get_Conversation

Get the current conversation and its attributes.

Get_TP_Properties

Get TP properties (TP name, LU name, user ID, profile).

Get_Type

Get the type of conversation (basic or mapped).

Post_on_Receipt

Receive asynchronous notification when data or status on particular conversation becomes available.

Prepare_to_Receive

Prepare to receive data (change to Receive state).

Receive_Immediate

Receive data immediately, including any buffered data.

Receive_and_Wait

Receive and wait for data to arrive.

Request_to_Send

Request to send data (change to Send state).

Send_Data

Send data to the partner.

Send_Error

Send error notification to the partner when the caller detects an error.

Set_Syncpt_Options

Change the APPC/MVS defaults that control syncpoint processing for protected conversations.

For reference information about these services, including syntax and parameter descriptions, see [Chapter 8, “APPC/MVS TP Conversation Callable Services,”](#) on page 125.

The following sections describe how to use these services in a TP conversation.

Starting a Conversation

To start a conversation with a partner TP, a local TP calls the Allocate service. On the call to Allocate, specify the conversation type (basic or mapped) and the destination (a symbolic destination name or explicit values for the partner TP name, LU name, and session mode). Also specify whether or not the conversation will include confirmation requests (a *sync_level* of *none* or *confirm*).

To protect against unauthorized conversations, Allocate must include the security information that the partner expects. Specify which of the following levels of security information you are providing:

Security_none

Specifies that no user ID, password, or profile (RACF group name) is passed for verification.

Security_same

Specifies that the same user ID and possible profile name that were used to allocate the local program are being passed on the current allocate call.

Security_pgm

Specifies that a user ID, password, and profile name are being provided for the partner LU to verify. If surrogate authorization is used, only the user ID is required. For more information, see [z/OS MVS Planning: APPC/MVS Management](#).

Asynchronous notification

On the Allocate call, you can specify an ECB to be posted when the service is complete. See [“Using Asynchronous Services”](#) on page 52 for more information about asynchronous notification for this and other services.

Authorized parameters

If your TP is running in supervisor state or PSW key 0-7, it can also specify the following on Allocate:

- A RACF user token (UTOKEN) to dictate the partner's security environment
- The TP_ID to be associated with the conversation.

Accepting a Conversation

When an inbound allocate request arrives for a TP under APPC/MVS control, APPC/MVS verifies that the request passes any security requirements for the partner LU and TP, and accepts or rejects the request accordingly. If accepted, APPC/MVS forwards the request to the transaction scheduler that initiates the partner TP. The partner TP must call the `Get_Conversation` service before any other APPC service to obtain information about the conversation. That returned information includes the conversation ID, conversation type, LU name, mode name, and `sync_level`. (If you are using an APPC/MVS server, you can use the `Receive_Allocate` service to retrieve the oldest allocate request from a particular allocate queue. See [z/OS MVS Programming: Writing Servers for APPC/MVS](#) for more information about the `Receive_Allocate` service.)

Obtaining Information about the Conversation

After accepting the conversation, the partner TP can then use the conversation ID as input to these services to obtain more information about the conversation:

Get_Type

Tells the caller what type of conversation is in effect (basic or mapped). This service allows a TP to handle requests from callers who might specify either type of conversation. The TP can use conditional logic to send and receive basic or mapped data depending on the type in use.

Get_Attributes

Tells the caller the LU, mode name, `sync_level`, the conversation correlator, and other information that applies to the caller in the current conversation. This allows the TP to run under different combinations of the above, using conditional logic to function appropriately. For example, a partner TP can use the conversation correlator, which uniquely identifies a conversation, to correlate requests with responses from an alternate transaction scheduler.

When a partner TP allocates a conversation with a TP running on MVS, the system on which the partner TP is running generates the conversation correlator, and sends the correlator to APPC/MVS in the FMH-5 for the `Allocate` request. After the scheduler processes the request, the scheduler sends the correlator back to the partner TP as data over the conversation. The partner TP can then correlate the original request with the response from the scheduler.

Note: When MVS is the partner system, it only generates unique conversation correlators for syncpoint conversations. For all others, MVS always provides a conversation correlator that has the value zero.

[Figure 17 on page 49](#) shows an example of how a scheduler can process a conversation correlator. The figure illustrates the following scenario:

1. The partner TP allocates a conversation with a local TP on MVS. The partner system sends the conversation correlator to APPC/MVS in the FMH-5 that contains the request.
2. The partner TP calls the `Send_Data` service to put a block of data in a buffer for sending.
3. The partner TP calls the `Get_Attributes` service to receive the conversation correlator assigned to the conversation.
4. The partner TP calls the `Deallocate` service to end the conversation and force the LU to send the buffered data.
5. The transaction scheduler receives the request and schedules the transaction. The scheduler receives the correlator from the IXCMGSI message buffer.
6. The scheduler allocates a conversation back to the partner TP.
7. The scheduler sends the conversation correlator to the partner TP as data over the conversation. The scheduler also sends the message that results from the execution of the TP.
8. The scheduler calls the `Deallocate` service, with a `Deallocate_type` of `Confirm`, to end the conversation and force the LU to send the buffered data.
9. The partner TP receives the conversation correlator.

10. The partner TP validates that the conversation correlator that it is the same as the correlator returned on the call to `Get_Attributes` in step “3” on page 48 above.
11. The partner TP calls the Confirmed service to send a confirmation reply to the transaction scheduler.

In Figure 17 on page 49, “CC” indicates the conversation correlator.

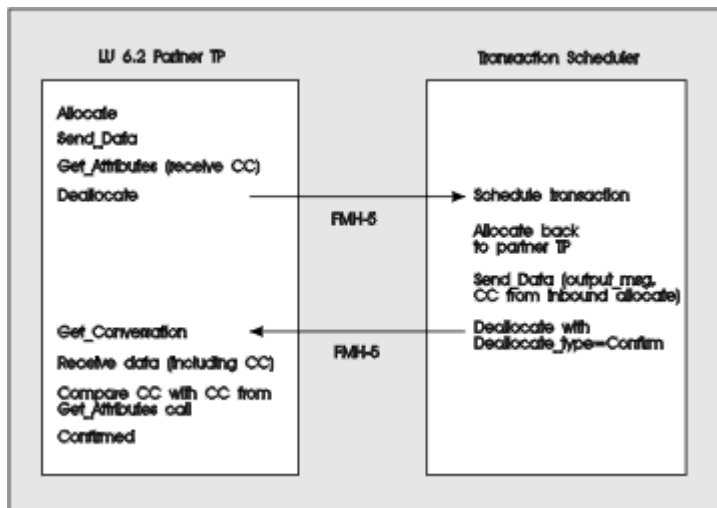


Figure 17. Example Use of the Conversation Correlator

Getting Current TP Properties

A newly allocated TP can call the `Get_TP_Properties` service to get more information about itself, including:

- Own TP name
- Own LU name
- User ID passed on the allocate request
- Profile passed on the allocate request
- LUW_ID
- Options for syncpoint processing that are in effect for protected conversations.

`Get_TP_Properties` is useful when a TP might run in different environments, for example, by being defined in different TP profiles to different MVS LUs, possibly under different TP names. `Get_TP_Properties` lets such a TP retrieve the name and LU under which it is running in a given instance. `Get_TP_Properties` also returns the LU work identifier (LUW_ID) associated with the TP, and the user ID and profile that account for the TP's current security environment.

Changing Syncpoint Options for Protected Conversations

When your installation enables APPC/MVS support for protected conversations, certain system defaults are in effect for each TP and its protected conversations. These defaults govern the way APPC/MVS interacts with other resource managers, and the system syncpoint manager (RRS), to coordinate resource recovery. This interaction adheres to the two-phase commit protocol, which is described in [z/OS MVS Programming: Callable Services for High-Level Languages](#).

Depending on the processing your TP and its partner TPs perform, you might want to change these default values. For example, you might change the value of the `Wait_For_Outcome` option. The default for this option is NO; changing it to YES allows a syncpoint operation to complete more quickly in the event of a session failure. The potential disadvantage is that, after your TP issues a `Commit` or `Backout` call, APPC/MVS might return control before it knows the outcome of the syncpoint operation.

An APPC/MVS TP, server, or alternate transaction scheduler may change syncpoint option values through a call to the Set_Syncpt_Options service. “Set_Syncpt_Options” on page 248 describes the options you may change, their default values, and other requirements for successfully calling this service.

Sending and Receiving Data

When a conversation is established, the local (calling) program enters Send state and its partner enters Receive state. The programs can then use the following services to send and receive data:

- Send_Data
- Receive_and_Wait
- Receive_Immediate

Support for Data Spaces

The above services allow data buffers to reside in MVS data spaces. To designate a data space, provide an access list entry token (ALET) along with the buffer address to designate the data space in which the buffer resides. APPC/MVS accepts any ALET that:

- Is a public ALET in the dispatchable unit access list (DU-AL) of the work unit that made the request
- Does not designate the secondary address space.

Parameter lists and parameters other than data buffers must reside in the caller's primary address space.

A sending program can use the send_type parameter to send information and data, and to combine operations, thus saving extra calls to APPC/MVS. Possible values of send_type are:

Buffer_data

Tells APPC/MVS to put the data in its own buffers until a sufficient quantity is accumulated.

Send_and_flush

Tells APPC/MVS to send the data immediately without buffering it.

Send_and_confirm

Tells APPC/MVS to send the data immediately along with a request for confirmation.

Send_and_prepare_to_receive

Tells APPC/MVS to send the data immediately along with send control, to put the partner in Send state.

Send_and_deallocate

Tells APPC/MVS to send the data immediately along with a notification that the conversation is deallocated.

The receiving program learns about the specified send_type and other requests for action by a value returned in the status_received parameter on its latest Receive_Immediate or Receive_and_Wait call:

Send_received

Indicates that the partner has entered Receive state, placing the local program in Send state.

Confirm_received

Indicates that the partner has requested a confirmation.

Confirm_send_received

Indicates that the partner has called the prepare_to_receive function and requested confirmation.

Confirm_dealloc_received

Indicates that the partner has called the deallocate function and requested confirmation. For syncpoint conversations the following can be received: TAKE_COMMIT, TAKE_COMMIT_SEND, and TAKE_COMMIT_DEALLOCATE.

The receiving program can specify a maximum amount of data to be received and, if the conversation is basic, whether to fill the buffer to the maximum or to the last complete logical record that fits. When data arrives, APPC returns a data_received parameter to tell the receiver if the buffer contained an incomplete record.

Requesting Permission to Send

TPs in Receive state can call `Request_to_send` to request permission to send data, for example, in response to something they received. APPC/MVS notifies the partner TP by passing `request_to_send_received` as a returned parameter on the partner's next Send, Receive, or Confirm call to APPC/MVS.

Granting Permission to Send

When a TP in Send state receives `request_to_send_received` as a returned value on one of its calls to APPC/MVS, the TP can ignore the request, or grant the request by calling one of the following services:

- `Receive_and_Wait`
- `Prepare_to_Receive`

Any one of the above services puts the caller in Receive state and its partner in Send state. Unlike `Receive_and_Wait`, `Prepare_to_Receive` does not require a receive buffer as input.

Requesting Confirmation

When a conversation was allocated with confirmation allowed (`sync_level` was set to *confirm* or *syncpt*), a sending TP can request confirmation, either on the `Send_Data` call (with `send_type` set to *Send_and_confirm*) or by calling the `Confirm` service. The partner is notified of the confirmation request by a value returned in the `status_received` parameter on its next receive call. The partner can then provide the confirmation by calling the `Confirmed` service, or reject the confirmation by calling the `Send_Error` service, the `Deallocate Type(Abend)` service, or (if the conversation is *syncpt*) with a Backout request.

Returning Error Information

When APPC/MVS finds an error in an APPC/MVS conversation, it returns an error return code. Because APPC/MVS return codes are "architected", meaning that they are common to every system that can participate in a SNA network, they provide only general descriptions of errors.

To obtain more detailed descriptions of errors that occur in APPC/MVS conversations, your application can call the `Error_Extract` service. `Error_Extract` returns reason codes and messages that further describe errors indicated by APPC/MVS return codes. If your TP is involved in a conversation with a TP that is running on a partner system, `Error_Extract` returns error log information and a product set identifier, if the partner system does all of the following:

- Supports the error log function (for more information, see the section on error log variables in [z/OS Communications Server: SNA Programmer's LU 6.2 Guide](#))
- Detects an error in a conversation with an APPC/MVS TP
- Sends error log information to APPC/MVS.

See Chapter 6, "Diagnosing Problems with APPC/MVS TPs," on page 77 for information about how to use the `Error_Extract` service.

Sending Error Notification

When a TP encounters an error in its processing or cannot grant a requested confirmation, the TP can call the `Send_Error` service to notify its partner. A TP can call `Send_Error` from Send or Receive state. The partner receives notice of the error by return code on its next call to APPC/MVS.

If the TP that detects the error is in Send-pending state (has just received data and send control on the same call), the TP needs to specify where it found the error: in data that it received from the partner or in data it was preparing to send. To do so, specify a value of *receive_error* or *send_error* in the `error_direction` parameter of the `Send_Error` service.

Ending Conversations

When your TP has finished communicating with its partner, deallocate the conversation by calling the Deallocate service, or the Send_Data service with a send_type of send_and_deallocate. On the deallocate_type parameter of the Deallocate service, specify one of the following:

Deallocate_sync_level

Deallocates based on the sync_level set by the allocate function:

- If sync_level is *none* (no confirmation), performs the function of the flush service and deallocates the conversation normally
- If sync_level is *confirm*, requests a confirmation; if granted, deallocates the conversation normally.
- If sync_level is *syncpt*, defers the deallocation until the program issues a Commit call. Upon successful completion of the Commit call, the system deallocates the conversation.

Deallocate_flush

Performs the function of the flush service, and deallocates the conversation normally.

Deallocate_confirm

Requests a confirmation:

- If granted (Confirmed), deallocates the conversation normally
- If rejected by Send_Error, the conversation continues.

Deallocate_abend

Performs the function of the flush service and deallocates the conversation abnormally.

For more information about the deallocate types, see the explanations for return codes 17 and 18 under Appendix B, “Explanations of Return Codes for CPI Communications Services,” on page 391. The partner is notified of the deallocate request by status_received parameter or by return code.

As a result of the deallocate function, both partners go into Reset state and the conversation ends.

Using Asynchronous Services

TPs can use most of the APPC/MVS TP conversation services asynchronously by specifying the address of an event control block (ECB) on the Notify_Type parameter. If APPC/MVS accepts the request for asynchronous processing, it gives a return code of zero for the service, and the TP can continue other processing without waiting for the service to complete. When all parameters are returned and the service completes, APPC/MVS notifies the TP by posting the ECB.

By contrast, when you call an APPC/MVS service *synchronously* (by setting the Notify_Type parameter to a value of *none*), your TP waits for the service to complete. Your TP regains control when APPC/MVS passes a return code and any other returned parameters from the service.

All conversation services available on synchronous calls are available on asynchronous calls as well.

When specified on the Notify_Type parameter, the ECB must be cleared to zero and meet all requirements for the POST macro. When the ECB is posted to indicate that asynchronous processing is complete, the completion code in the ECB is the return code for the service. All input parameters, except for data buffers, are processed before return to the TP, so that the TP is free to use these areas on return without affecting the asynchronous processing. However, data buffers passed as input, and all output parameters, are accessed and manipulated directly by the asynchronous processing, and therefore should not be referenced or modified by the TP until it has been notified that call processing is complete.

APPC rejects any service call from a conversation for which an asynchronous call was previously issued and did not complete. The only exceptions are Deallocate (ATBDEAL) calls with a deallocate_type of ABEND. Those calls are accepted when the conversation has an outstanding asynchronous call.

The asynchronous capability is recommended for the following types of APPC/MVS transaction programs:

- Multiprocessing transaction programs
- Transaction programs that cannot afford to be suspended

- Transaction programs processing multiple conversations in a single dispatchable unit.

The asynchronous capability is available through the `Notify_type` parameter on the following TP conversation services:

- `Allocate_Conversation`
- `Confirm`
- `Confirmed`
- `Deallocate_Conversation`
- `Flush`
- `Prepare_to_Receive`
- `Receive_and_Wait`
- `Request_to_Send`
- `Send_Data`
- `Send_Error`

Invoking an authorized command or program in a TSO/E session while any asynchronous notifications from APPC/MVS callable services are outstanding is not supported. In addition, asynchronous APPC/MVS calls are not supported across a job step boundary.

Using the Asynchronous_Manager Service

You can call the `Asynchronous_Manager` service to determine whether there are any asynchronous APPC/MVS calls outstanding in an address space, and if necessary, to deallocate any conversations that have asynchronous calls outstanding. For more information, see [“Identifying and Deallocating Conversations with Outstanding Asynchronous Requests”](#) on page 64.

Obtaining Asynchronous Notification of Data to be Received

To be notified asynchronously when data or status is ready to be received on a conversation, use either of the following services:

- `Post_on_Receipt`
- `Receive_and_Wait` (with the `Receive_Length` parameter set to zero).

APPC/MVS posts an ECB specified by the caller when data or status, or both, is available to be received.

The advantage of using `Post_on_Receipt` is that, unlike `Receive_and_Wait`, it does not tie up the conversation while it is processed. The caller is free to call other conversation services on the same conversation (such as `Request_to_Send`) while APPC/MVS processes the `Post_on_Receipt` request asynchronously. `Post_on_Receipt` can be called on basic conversations only.

The advantage of `Receive_and_Wait` is that it returns more information (such as the `Status_Received` or `Data_Received` parameters) than `Post_on_Receipt`. With `Post_on_Receipt`, you must follow the call with a call to `Receive_and_Wait` or `Receive_Immediate` to obtain this information. `Receive_and_Wait` can be called on both basic and mapped conversations.

Both methods are described in the following sections.

Using Post_on_Receipt

`Post_on_Receipt` notifies the caller (through an ECB the caller specifies) for the following situations:

- A complete logical record is available to be received from the partner program
- Conversation status (control information) is available
- A nonzero return code is available to be received because of an action taken by the partner program (such as deallocating the conversation).

When Post_on_Receipt posts the specified ECB, the caller can determine which of the preceding information is available by calling the Receive_and_Wait or Receive_Immediate service and checking the returned parameters.

The following is a pseudocode example of calling the Post_on_Receipt service:

```
<issued from receive state>

ECB=0                                /* ECB cleared to zero      */

CALL ATBPOR2
    (Conv_ID,                        /* In - Conversation ID      */
     ECB_Address,                   /* In - Address of ECB       */
     Return_Code);                 /* Out - Return Code        */

If Return_Code = 0 THEN
    <do other work, such as call Request_To_Send>
    ...
    WAIT for ECB to be posted...
```

Figure 18. Obtaining Asynchronous Notification With Post_on_Receipt

The ECB specified by the ECB_Address parameter is posted when data or status, or both, is available to be received.

Your call to Post_on_Receipt remains in effect until the specified ECB is posted, or the call is cancelled. Thereafter, to obtain subsequent notification of data or status to be received, issue a new call to the Post_on_Receipt service.

Using Receive_and_Wait with a Receive_Length of Zero

To be notified asynchronously when there is data ready for you to receive on a conversation, you can also use an asynchronous Receive_and_Wait call with a Receive_Length of zero. The following is a pseudocode example of calling the Receive_and_Wait service, with the key parameters indicated by arrows:

```
DECLARE
    Conv_Id CHAR(8),                /* Conversation ID           */
    Fill_Value FIXED,               /* Fill Value                */
    Receive_Length BIT(32),         /* Receive Length            */
    Buffer_ALET BIT(32),            /* Buffer Access Token        */
    Dummy_Buffer CHAR(1),          /* Dummy buffer              */
    Status_Received FIXED,         /* Status Received           */
    Data_Received FIXED,           /* Data Received             */
    Req_to_Send_Received FIXED,    /* Request to Send Received  */
    My_Ecb FIXED,                  /* ECB to post               */
    Return_Code FIXED;             /* Return code               */

DECLARE 1 Notify_Value,            /* Eight-Byte structure      */
        2 Notify_Type FIXED,      /* Asynchronous notify type  */
        2 My_Ecb_Address PTR(31); /* Address of My_Ecb        */

Receive_Length = 0;               /* Initialize Receive Length */
Notify_Type = atb_notify_type_ecb; /* Asynchronous notification */
My_Ecb_Address = ADDR(My_Ecb);    /* Get address of ECB        */

CALL ATBRCVW
    (Conv_Id,                      /* In - Conversation ID      */
     Fill_Value,                   /* In - Fill Value          */
     <====> Receive_Length,         /* In-out - Receive Length  */
     Buffer_ALET,                  /* In - Buffer ALET          */
     <====> Dummy_Buffer,          /* Out - Buffer              */
     Status_Received,             /* Out - Status Received    */
     Data_Received,              /* Out - Data Received      */
     Req_To_Send_Received,        /* Out - Req To Send Received */
     <====> Notify_Value,         /* In - Notify type         */
     Return_Code);               /* Out - Return Code        */
```

Figure 19. Obtaining Asynchronous Notification With Receive_and_Wait

The ECB specified by the Notify_Value parameter is posted when data, status, or both, are available to be received. No buffer need be specified, because no data is returned on this call (you can specify a "dummy" variable). When the ECB is posted, the output parameters are filled in (Status_Received,

Data_Received, Req_To_Send_Received, and Return_Code). If there is data to be received, you can issue a Receive_Immediate or another Receive_and_Wait call to receive the data. This subsequent call could be with CPI Communications as well.

To obtain subsequent notifications of data to be received, issue a new asynchronous Receive_and_Wait call after each receipt of data.

Setting a Timeout Value for Potential Network Delays

APPC/MVS uses the VTAM APPCCMD macro to communicate with partner applications. Sometimes a VTAM APPCCMD request does not complete, for example because of a network delay or a delay at the partner LU. If this happens, the APPC/MVS transaction program can hang while waiting for control to be returned by APPC/MVS. By setting a time limit for VTAM APPCCMD requests, you can avoid such hangs and regain control of the conversation.

If the time limit is reached before the VTAM APPCCMD request completes and returns control to APPC/MVS, the conversation will be terminated by APPC/MVS and the caller of the conversation callable service will regain control. If the conversation call was issued with a Notify_Type=ECB (asynchronous processing), the specified ECB will be posted when the time limit is reached.

To set a time limit, use the Set_TimeOut_Value service or either the Timeout_Value_Minutes or Timeout_Value_Seconds parameter on the new version of the Allocate service (ATBALC6).

You can invoke the Set_Timeout_Value service any time after the conversation is successfully established. You can also invoke the Set_Timeout_Value service to alter a previously set timeout value. All subsequent APPC/MVS or CPIC callable services will be limited to the timeout values specified on the Set_Timeout_Value or allocate call.

At any time your program wishes to query the timeout value that is currently set for a conversation, use the Get_Attributes service (ATBGTA6). The combined values, in the form of seconds, of the Timeout_Value_Minutes and Timeout_Value_Seconds will be returned.

You can use the Version_Service to determine whether Set_Timeout_Value or the new version of the Allocate service is available on the current system. An application can invoke Set_Timeout_Value or a version of Allocate which supports specifying the timeout value when the Callable_service_version_number returned from the Version_Service is 5 or higher. With a version number of 5, the application specifies the value in minutes. If the value is 6 or higher, the application specifies the timeout value at the minute and/or second level, providing greater granularity in the timeout value specified.

If an APPC/MVS or CPIC callable service is interrupted because a timeout limit was reached, the interrupted conversation call will return one of the following return codes:

- For syncpoint conversations: Resource_Failure_Retry_BO return code or, if a Resource_Failure_Retry_BO return code is not possible, a Product_Specific_Error return code.
- For other conversations: Resource_Failure_Retry or, if a Resource_Failure_Retry return code is not possible, a Product_Specific_Error return code.
- For the Deallocate service issued with the Deallocate_type parameter set to Deallocate_abend, the OK return code.

If a conversation is terminated during time-out processing, any subsequent APPC/MVS or CPIC callable service will fail with a Program_Parameter_Check return code.

If an error occurs, invoke the Error_Extract service to get a detailed description of the error. If API trace was started for the application, then a detailed error message will also be captured in the trace data set.

When a distributed transaction calls the Commit or Backout service, it enters a syncpoint processing phase. During this time, APPC effectively takes over the existing conversations which connect the nodes of the distributed transaction. APPC uses those conversations to exchange various syncpoint messages. Time-out processing is disabled for all message exchanges during syncpoint processing. A hang at one of the nodes during syncpoint processing will therefore hang the entire distributed transaction. For more

information, see [“ATBST05 - Set_TimeOut_Value \(For OS/390 Release 8 through z/OS V1R6\)”](#) on page 474.

Performance Considerations for Conversation Services

The relative performance of APPC/MVS callable services varies depending on the functions that the callable service performs. For example, services that call VTAM or cause the movement of data buffers involve a greater number of internal instructions. For an overview of performance considerations for individual APPC/MVS TP callable services, see [Table 5 on page 56](#).

<i>Table 5. Performance Considerations for MVS TP Callable Services</i>					
APPC/MVS Service	Calls VTAM	Causes DASD I/O	Causes buffer moves	Calls RACF	Creates SMF record
Allocate	Yes	Sometimes(1)	No	RACROUTE= TOKENXTR, TOKENMAP	No
Confirm	Yes	No	No	No	No
Confirmed	Yes	No	No	No	No
Deallocate	Yes	No	No	No	No
Error_Extract	No	No	No	No	No
Extract_Information	No	No	Yes	No	No
Flush	Yes	No	No	No	No
Get_Attributes	No	No	No	No	No
Get_Conversation	No	No	No	No	No
Get_TP_Properties	No	No	No	RACROUTE= TOKENXTR, TOKENMAP	No
Get_Transaction	No	No	No	RACINIT	SMF 33
Get_Type	No	No	No	No	No
Post_on_Receipt	No	No	No	No	No
Prepare_to_Receive	Yes	No	No	No	No
Receive_Immediate	Yes	No	Yes	No	No

<i>Table 5. Performance Considerations for MVS TP Callable Services (continued)</i>					
APPC/MVS Service	Calls VTAM	Causes DASD I/O	Causes buffer moves	Calls RACF	Creates SMF record
Receive_and_Wait	Yes	No	Yes	No	No
Register_Test	No	Yes (2)	No	RACROUTE= TOKENXTR	No
Reject_Conversation	Yes	No	No	No	No
Request_to_Send	Yes	No	No	No	No
Return_Transaction	No	No	No	RACINIT	SMF 33
Send_Data	Yes	No	Yes	No	No
Send_Error	Yes	No	No	No	No
Set_Conv_Accounting_Info	No	No	No	No	No
Set_Syncpt_Options	No	No	No	No	No
Unregister_Test	No	Yes (2)	No	RACROUTE= TOKENXTR	No
Note: 1. Might read from the side information file. 2. Modifies the TP profile.					

One general performance tip is to minimize the number of calls to APPC/MVS for conversation services. Try to design a transaction program in such a way that it causes the least amount of code execution in APPC/MVS as possible, without overly complicating the program.

Each time APPC/MVS is called, APPC/MVS checks to see if the caller's parameters contain valid values. When possible, combine functions in a single call, and send and receive large blocks of data. Some specific ways to do this are described in the following suggestions.

- Send as much data as possible on all Send_Data calls, thus minimizing the number of sends and receives.
- Use the type parameter on the Send_Data service to combine operations.

In APPC/MVS, a TP can combine other conversation operations, such as Flush, Confirm, and Deallocate, with each Send_Data call. This can often save an extra call to APPC.

- Minimize use of the Confirm service.

The Confirm service forces a transaction program to wait for an explicit acknowledgement from the partner. Use the confirm functions only when acknowledgement is required.

- Minimize use of the Request_to_Send and Prepare_to_Receive services.

When possible, the local and partner TPs should be written so that they are aware of the current state of the conversation, and exchange their Send and Receive states when appropriate. The state of the conversation is known at all times, based on what call was issued and the return codes and indicators that are returned. Appendix C, “APPC/MVS Conversation State Table,” on page 399 describes these states for each MVS TP service, indicator, and return code. *CPI-C Reference* describes these states for each CPI Communications call.

- Minimize use of the Get_Type, Get_Attributes, and Get_TP_Properties services.

Your TPs should know most of this information already. If these services are called, their returned values should be kept in local program variables for later use.

- Use the Flush service judiciously.

The Flush function forces APPC/MVS to send the data in its internal buffers, even though those buffers might not be full. Unless the partner needs the data immediately, let APPC/MVS determine when it is most efficient to send the data.

If your TP calls a Send_Data service and then plans on calling no other conversation services for a while, it should probably call the Flush service or Send_Data with Send_Type of Flush to clear its buffers, so the data can be used by the partner TP.

- Send one logical record per Send_Data call.

In basic conversations, send as much data as possible on each send, but use only one LL (record length field) per call. Avoid splitting LL fields across calls.

- Receive as much data as possible on each Receive call.

Allocate a large storage area for incoming data, and receive as much data as possible on each Receive_and_Wait or Receive_Immediate.

In basic conversations, specify a value of Buffer for the Fill parameter on Receive_and_Wait and Receive_Immediate, instead of a value of LL.

- Use the deallocate_type of deallocate_flush on Deallocate calls.

The deallocate_type of deallocate_flush is faster than the other types on the Deallocate service.

Advanced TP Services

In addition to conversation services, APPC/MVS provides advanced services that you can use to extend, customize, and test TP processing. The advanced services are specific to MVS and have no LU 6.2 or CPI-C equivalents. The advanced services allow you to:

- Extract detailed scheduling and conversation information about a TP
- Add user data to SMF accounting records
- Use multi-trans scheduling to accept multiple conversations in sequence
- Reject inbound conversations if needed
- Test transaction programs in a non-APPC address space.

The following sections describe how to use these advanced TP services. For reference information, including syntax and parameter descriptions, see [Chapter 9, “APPC/MVS Advanced TP Callable Services,” on page 259.](#)

Extracting Detailed Scheduling and Conversation Information

With the Extract_Information service, APPC/MVS transaction programs can extract detailed information about their own conversations and scheduling. If in supervisor state or PSW key 0-7, they can extract conversation information about other TPs.

You can call the Extract_Information service from a TP to extract scheduling information such as the transaction scheduler, the transaction initiator class, and the TP schedule type under which the TP is running. You can also extract conversation information, such as the total number of conversations in

which the TP is involved, the number of sends and receives it has performed, the unit of recovery identifier (URID) for a protected conversation, and the total amount of data sent and received.

TPs can save this extracted information in a data set for later analysis or use it dynamically. For example, a TP can use the `Extract_Information` service to check whether it is scheduled with the multi-trans schedule type and, if so, call the `Get_Transaction` service to accept new conversation requests in sequence.

As input to the `Extract_Information` service, specify the type of information you want to extract (scheduler, summary conversation, or specific conversation), the TP or conversation involved (the caller's or, if the caller is authorized, the `TP_ID` or `Conv_ID` of another TP or conversation whose information you want), and a buffer for the output. The buffer can be ALET-qualified, subject to the limits described in [“Support for Data Spaces”](#) on page 50.

To access mappings of the information returned in the buffer that can be used by assembler language programs, include the following mapping macros in the calling TP:

ASBEXSCH

For scheduling information

ATBEXCON

For summary conversation information

ATBEXCOS

For specific conversation information

Adding User Data to Accounting Records

When either partner deallocates an APPC/MVS conversation, SMF writes a type 33, subtype 2 record. SMF provides detailed information about both partners, their associated LUs, and the amount of data transferred over the network.

TPs can write up to 255 bytes of user-defined data to these accounting records through the `Set_Conversation_Accounting_Information` service. The user data also appears in the user data field that the `Extract_Information` service returns when TPs extract information about specific conversations.

You can use the user data field to charge resources to a specific conversation, or to correlate outbound conversations to inbound conversations, perhaps by specifying a conversation ID in the user data.

Using TP Schedule Types

The APPC/MVS transaction scheduler lets you assign transaction programs to a schedule type of **standard** or **multi-trans**.

Using the Standard Schedule Type

When transaction programs are scheduled as standard (the default), APPC/MVS initializes them for each inbound conversation request and ends them when they finish processing. A transaction program scheduled as standard runs as shown in [Figure 20 on page 60](#).

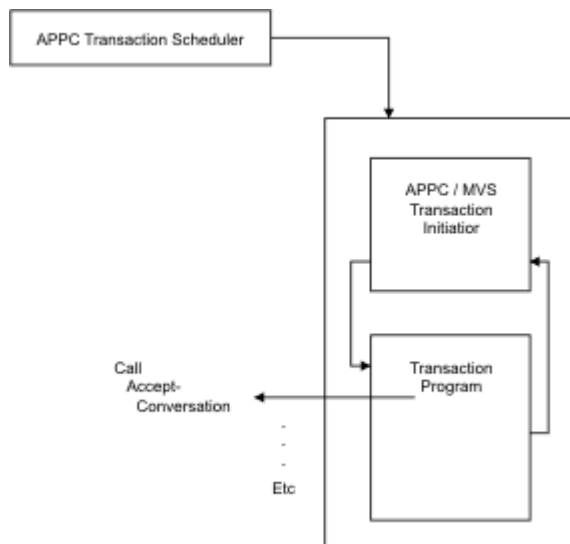


Figure 20. Standard Scheduling for an APPC/MVS Transaction Program

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 that are not invoked very frequently or do not require extensive allocation of resources.

Using the Multi-Trans Schedule Type

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

Figure 21 on page 61 shows how a multi-trans program receives and processes consecutive transaction requests. The multi-trans program is typically coded within a *multi-trans shell*, an environment that performs initialization and termination processing, surrounding the part of the TP that holds conversations. When a multi-trans program is scheduled in response to an initial allocate request, the multi-trans shell gets control first and allocates general resources, then calls the `Get_Transaction` service to obtain the inbound request. When the conversation ends, the shell regains control and calls the `Get_Transaction` service again when the transaction program is ready to handle the next request. Use of a loop structure lets you do the same processing for each requesting user.

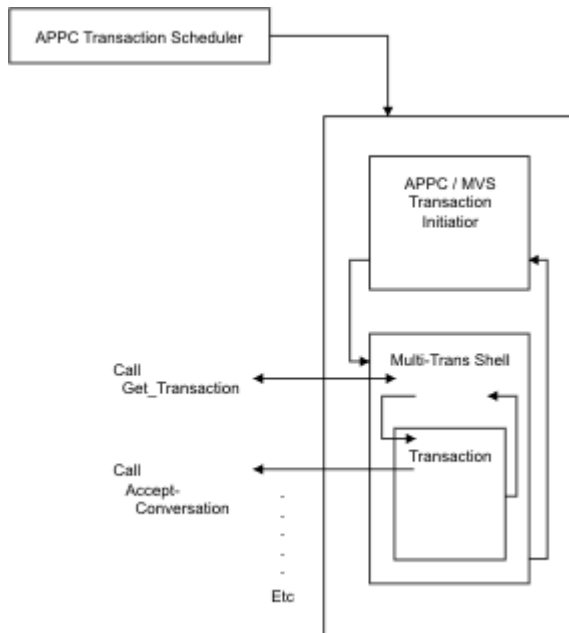


Figure 21. Multi-Trans Scheduling for an APPC/MVS Transaction Program

To use multi-trans scheduling, transaction programs must be defined with a schedule type of multi-trans in the TP profile. The TP profile entry must also contain a user ID to provide the initial security environment for the multi-trans shell. The shell runs under this *generic user ID* during initialization, while the shell allocates general resources for the TP to use. The generic user ID remains in effect until the first successful `Get_Transaction` call, when the security environment is personalized to the user ID associated with the inbound request. That personalized security environment covers the entire conversation and remains in effect until the next `Get_Transaction` call, or until the shell explicitly returns to its generic security 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` 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 several individual IDs.

The multi-trans shell is responsible for doing all necessary cleanup between conversations to ensure the integrity and security of the TP's conversations and data for consecutive users. The shell can do the cleanup under the same user ID as the preceding conversation, or the generic user ID.

When there are no more inbound requests for a multi-trans TP to process, control returns under the generic user ID. A return code from `Get_Transaction` indicates whether or not the caller can invoke the `Get_Transaction` service again to wait for more work to arrive. If the shell does not call `Get_Transaction` again, it should clean up any remaining resources and end the program.

Figure 22 on page 62 shows the different phases of multi-trans processing. Shaded sections are covered by the generic user ID; the rest is under personalized security.

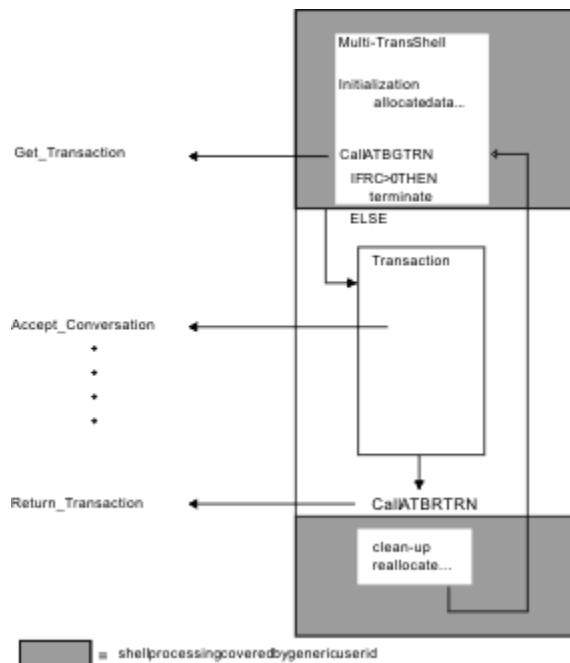


Figure 22. Phases of Multi-Trans Processing

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

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 transactions that are requested often by multiple users, that have a high resource overhead, and that finish processing comparatively quickly. For example, multi-trans processing could be appropriate for a phone book application that brings an entire phone book into a data space, then supplies phone numbers to multiple users in sequence.

Examples of Multi-Trans Scheduling

The following are some pseudocode examples of TPs that use multi-trans scheduling. When the TP is initiated, the shell code gets control first, calls `Get_Transaction` to obtain the first request, and repeats the call to get subsequent transactions. For example:

```
initialization for all users
CALL ATBGTRN to obtain work request
DO WHILE WORK_IS_REQUESTED
  :
  transaction processing for the user
  :
  CALL ATBGTRN to obtain the next work request
END WHILE
termination
```

You can code a transaction program to function as either standard or multi-trans, depending on the schedule type specified in the TP profile. To determine which TP schedule type it was invoked with, a transaction program can use the information extract service (see [“Extract_Information” on page 266](#)), and then do the appropriate processing. For example:

```
(extract TP information -- call ATBEXAI)
IF TP_IS_SCHEDULED_AS_A_STANDARD THEN
  initialization
  transaction processing for the user (TP scheduled as STANDARD)
  termination
ELSE
  initialization for all users
```

```

CALL ATBGTRN to obtain work request
DO WHILE WORK_IS_REQUESTED
:
:   transaction processing for the user
:
:   CALL ATBGTRN to obtain the next work request
END WHILE
termination
END IF

```

As an alternative to calling the Extract_Information service, the multi-trans shell can issue the Get_Transaction service in any case and continue with standard or multi-trans processing based on the return code from Get_Transaction. For example:

```

initialization (possibly for all users)
CALL ATBGTRN to obtain work request for a specific user
IF RC=16 (Not a multi-trans environment) THEN
:   transaction processing for the user (TP scheduled as STANDARD)
ELSE
DO WHILE WORK_IS_REQUESTED
:
:   transaction processing for the user
:
:   CALL ATBGTRN to obtain the next work request
END WHILE
END IF
termination

```

The multi-trans program can return to its shell environment by calling the Return_Transaction service between conversations, to clean up resources or allocate new ones if necessary. For example:

```

initialization for all users
allocate data sets
CALL ATBGTRN to obtain work request
DO WHILE WORK_IS_REQUESTED
:
:   transaction processing for the user
:
:   IF data sets are full THEN
:       CALL ATBRTRN to return to shell environment
:       Reallocate data set
:   CALL ATBGTRN to obtain the next work request
END WHILE
termination

```

Security for the Standard and Multi-Trans Schedule Types

Multi-trans programs should be trusted applications. They must do whatever cleanup is necessary between transactions to ensure that resources are released and transactions 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.

The generic user ID, specified in the TP profile, covers initialization and termination processing by the multi-trans shell, and any interim processing between a Return_Transaction and subsequent Get_Transaction call.

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 generic ID, you can use RACF to control read and update access to the TP profile where the generic ID is specified.

To do so, your installation can define the APPCMVS.TP.MULTI resource to the RACF FACILITY class and put the authorized administrators on the resource access list, as described in *z/OS MVS Planning: APPC/MVS Management*. The administration of multi-trans TP profiles must be limited to administrators

who have the same authority as the security administrator or to a system programmer who is responsible for updating authorized libraries.

Performance Considerations for TP Schedule Types

APPC/MVS offers potential performance benefits to transaction programs that are suited and properly designed for use with the multi-trans schedule type. To justify multi-trans scheduling, it must be less resource-intensive to change the user associated with the running of a transaction than to reinitiate the TP in an environment already set up for the particular user. Therefore, be sure to assess resource consumption before deciding on a TP schedule type.

Assigning Multi-Trans TPs to their own Class of Transaction Initiators

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 the ASCHPMxx parmlib member 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. See [z/OS MVS Planning: APPC/MVS Management](#) for more details about defining classes of transaction initiators.

Establishing a Multi-Trans Transaction Program that is Always Available

Multi-trans scheduling is especially suitable for APPC/MVS 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 of initiators.
2. In the TP profile, 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.

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

1. Set its TP profile inactive to prevent new inbound requests
2. Allow it to process any queued requests
3. Activate the TP profile for the new multi-trans TP.

SMF Recording for Multi-Trans Services

The Get_Transaction and Return_Transaction services create SMF records. For performance reasons, if records are not wanted, you can set SMF inactive on the system using the SMFPRMxx PARMLIB member. For more information about SMF recording, see [z/OS MVS System Management Facilities \(SMF\)](#).

For reference information about multi-trans scheduling, including syntax and parameters, see [“Get_Transaction” on page 272](#) and [“Return_Transaction” on page 281](#).

Identifying and Deallocating Conversations with Outstanding Asynchronous Requests

In some situations, a program might need to determine whether an address space contains any outstanding asynchronous APPC/MVS calls, including calls to ATBxxxx services that have an asynchronous processing specified on the Notify_Type parameter. For example, an authorized command or program cannot be invoked from a TSO/E address space while an asynchronous APPC/MVS call is outstanding.

In such situations, you can call the Asynchronous_Manager service to determine whether asynchronous APPC/MVS calls are outstanding, and if necessary, clean up the conversations that hold them. For example, when TSO/E needs to invoke an authorized command or program, TSO/E calls the Asynchronous_Manager service to ensure that no asynchronous calls are outstanding.

Rejecting Conversations

When a TP determines that it should not process a particular conversation it has accepted, the TP can reject the conversation by calling the `Reject_Conversation` service. For example, a TP might reject a conversation if it determines that the conversation has an unsupported `sync_level`. When it rejects a conversation, the TP communicates the reason for the rejection to its partner by specifying a sense code as an input parameter to the `Reject_Conversation` service. The partner LU resolves the sense code to a return code that it passes it to the partner TP.

The `Reject_Conversation` service must be used early in a transaction; that is, before any communication activity has occurred (such as data being sent or received).

Testing TPs

Programs can use TP test services to act as test shells for new or modified transaction programs that are not ready to run on the system. The test services let you run a TP in a non-APPC address space and possibly debug the TP interactively. The test services are:

Register_Test

Registers a TP for testing in the user's address space.

Accept_Test

Accepts a TP for testing and waits for its partner to request it.

Unregister_Test

Cancels a pending TP test.

Cleanup_TP

Deallocates and cleans up a test conversation.

For information about using these test services, see [Chapter 5, “Installing and Testing Transaction Programs,”](#) on page 67.

System Services

APPC/MVS provides authorized callable services that system programmers can use to add their own transaction schedulers to the one provided by APPC/MVS. For details about the system services that APPC/MVS provides, see *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

Example APPC/MVS Transaction Programs

The `SYS1.SAMPLIB` and `SYS1.MACLIB` data sets contain examples of APPC-related interfaces, such as parmlib members, JCL, and TPs written in Assembler, C, COBOL, FORTRAN, REXX, and RPG.

The sample TPs show how to call APPC/MVS services from programs that run under APPC/MVS. For instructions on using any of the pairs of example TPs, see the comments at the beginning of the outbound TP in that pair.

In past editions of this document, some of the example interfaces provided in `SYS1.SAMPLIB` were listed in this chapter. Those examples were removed because they only duplicated information that can be found in `SYS1.SAMPLIB` and other documents in the APPC/MVS library. [Table 6 on page 65](#) shows where you can find those examples now:

Table 6. Where to Get Examples of APPC/MVS Interfaces	
Description	Location
List of all the APPC-related materials in <code>SYS1.SAMPLIB</code>	<code>SYS1.SAMPLIB</code> member <code>ATBALL</code>
List of all the APPC-related materials in <code>SYS1.MACLIB</code>	Table 7 on page 66

Table 6. Where to Get Examples of APPC/MVS Interfaces (continued)

Description	Location
Example TPs, written in PL/I: <ul style="list-style-type: none"> • PL/I TP, pair 1, outbound standard • PL/I TP, pair 1, inbound standard • PL/I TP, pair 2, outbound multi-trans • PL/I TP, pair 2, inbound multi-trans 	SYS1.SAMPLIB member: <ul style="list-style-type: none"> • ATBOUTP1 • ATBINTP1 • ATBOUTP2 • ATBINTP2
Example jobs to create standard and multi-trans TP profiles	"Program Management" chapter in z/OS MVS Planning: APPC/MVS Management

Table 7 on page 66 lists the APPC/MVS-related materials provided in SYS1.MACLIB.

Table 7. List of APPC/MVS-Related Materials in SYS1.MACLIB

SYS1.MACLIB Member	Description
ATBAPPCA	APPC Component Control Block– Mapping of APPC/MVS-specific information that is available for use by the installation
ATBASASM	Pseudonym file for APPC/MVS Version_Service– Assembler Services
ATBCSASM	Pseudonym file for APPC/MVS Callable System Services– Assembler
ATBCTASM	Pseudonym file for APPC/MVS Callable Transaction Services– Assembler
ATBDFTP	APPC SDFM Transaction Profile (TP) Key Mapping Macro and TP Mapping Macro
ATBDFTPE	APPC SDFM TP Profile Conversion Exit Parameter Control Block
ATBEXCON	APPC Extract Conversation Information Control Block Mapping
ATBEXCOS	APPC Extract Specific Conversation Information Control Block Mapping
ASBEXSCH	APPC extract scheduler information control block mapping macro; defines the layout of the data that the system returns after a program calls the Extract_Information service
ATBSECB	Maps the information passed to the Scheduler Extract Exit
ATBSERV	Pseudonym file for LU 6.2 Protocol Boundary Interface– Assembler
ATBXCFSM	Mapping of each APPC/XCF message in APPC, and constants for the APPC/XCF message types.

Chapter 5. Installing and Testing Transaction Programs

After you have written an APPC/MVS transaction program (TP), you need to install and test it on MVS. After you test the TP and fix any problems in the code, you can make the TP available to other users on the system.

This chapter describes how to:

- Install a TP for testing
- Test a TP on MVS
- Collect problem data for errors that occur when testing a TP
- Put a tested TP into production
- Replace an active TP with a new version of the TP.

Installing a TP for Testing

Before you test a TP, you must do the following to install the TP:

1. Compile or assemble the TP.
2. Load the TP in a load module, or link-edit the TP in a load module with one or more of the following modules from SYS1.CSSLIB:

ATBPBI

For TPs that call CPI Communications or TP conversation services.

ATBATP

For TPs that call advanced TP services.

ATBCTS

For TPs that call Reject_Conversation or Set_Conversation_Accounting_Information.

After installing the TP, you can test the TP using one of the methods described in the following section.

Testing a TP on MVS

The best way to test a TP on MVS is to:

- Test a private copy of the TP under a TSO/E user ID, then
- Test the TP under APPC/MVS scheduling (while restricting use of the TP to your own TSO/E user ID).

For these tests, you must create an environment called a *test shell*, which allows the TP to run in any address space as if it were running in an APPC/MVS initiator address space. You can also include interactive debugging facilities in the test shell to help test and diagnose the TP.

To create a test shell, you must call the APPC/MVS test services described in [“Descriptions of APPC/MVS Test Services”](#) on page 68. APPC/MVS provides three methods that you can use to invoke the test services, which are described in [“Methods You Can Use to Create a Test Shell”](#) on page 67.

Methods You Can Use to Create a Test Shell

You can use one of the following methods to test a TP on MVS:

Table 8. Methods For Testing TPs on APPC/MVS

Method	Description	Reference
APPCTEST Interface	A collection of panels, REXX procedures, and programs that allow you to call APPC/MVS test services from the "Foreground Selection Panel" under ISPF/PDF.	<i>APPC Application Examples in MVS/ESA and OS/2</i>
Direct calls to test services	A method by which your application can call APPC/MVS services directly. The test services divert the TP to a TSO/E address space.	“Calling APPC/MVS Test Services from Your Application” on page 69
TSO/E TEST command	A command that invokes APPC/MVS test services for assembler TPs only.	“Using the TSO/E TEST Command to Test an Assembler Language TP” on page 70

The APPCTEST interface has certain advantages over calling the APPC/MVS test services directly. For example, when using APPCTEST, you do not have to insert calls to APPC/MVS test services into your application code. Therefore, you can use the same application code during the test and production phases for your TP. See *APPC Application Examples in MVS/ESA and OS/2* for a detailed explanation of the advantages of using APPCTEST.

The following section describes the APPC/MVS test services that you can invoke using any one of the methods described earlier.

Descriptions of APPC/MVS Test Services

The following APPC/MVS test services are used to create a test shell for a TP under test:

Register_Test

Places an inbound TP (the TP to be tested) in "test mode", and performs the setup required to start the TP in the caller's address space when inbound allocate requests arrive for the TP.

Accept_Test

Informs the APPC address space that the test shell is ready to accept inbound allocate requests for the TP, and suspends execution on the issuing address space.

Unregister_Test

Cancels an outstanding Register_Test request to prevent the TP from running under the test shell.

Cleanup_TP

Deallocates and cleans up any conversations and resources that are left after the test is complete. You must call this service if you plan to create another test session in the same TSO/E address space.

Test Shell Characteristics

The required environment for a test shell TP is:

Minimum authorization:

Problem state, any PSW key

Dispatchable unit mode:

Task mode

Cross memory mode:

Any PASN, any HASN, any SASN

AMODE:

31-bit

ASC mode:

Primary or access register (AR)

Interrupt status:

Enabled for I/O and external interrupts

Locks:

No locks held

Control parameters:

All parameters must be addressable by the caller and in the primary address space.

Other requirements are:

- The test shell program must have a Schedule_Type of Standard (not Multi_Trans) in the TP profile. To test a TP that was written as multi-trans, comment out the sections of code that issue the Get_Transaction and Return_Transaction services.
- The user must have access to run the inbound TP.

Calling APPC/MVS Test Services from Your Application

To create a test shell by calling APPC/MVS test services from your application, do the following:

1. Logon to TSO/E with a logon procedure that does not allocate APPC/MVS conversations when an APPC/MVS TP is to be tested.
2. Catalog the data set that contains the load module for the TP to be tested.
3. Allocate all required data sets to the TSO/E address space where the TP is to run, including a data set for TP message log. See the section on logging transaction program processing in *z/OS MVS Planning: APPC/MVS Management*.
4. Ensure that no active APPC conversations are in progress in the TSO/E address space, and that all resources from prior conversations are cleaned up in the TSO/E address space.
5. Establish the required environment for the TP under test, as described in [“Test Shell Characteristics” on page 68](#).
6. Ensure that the TP under test belongs to an APPC/MVS local LU that allows user-level TP profiles. Although you are not required to have a user-level TP profile when testing a TP under a TSO/E user ID, you must enable the LU for user-level TP profiles. See [“Enabling an LU for User-Level TP Profiles” on page 73](#).
7. Enter calls to the Register_Test and Accept_Test services from the TP to be tested. The sequence of these calls is shown in [Figure 23 on page 70](#):

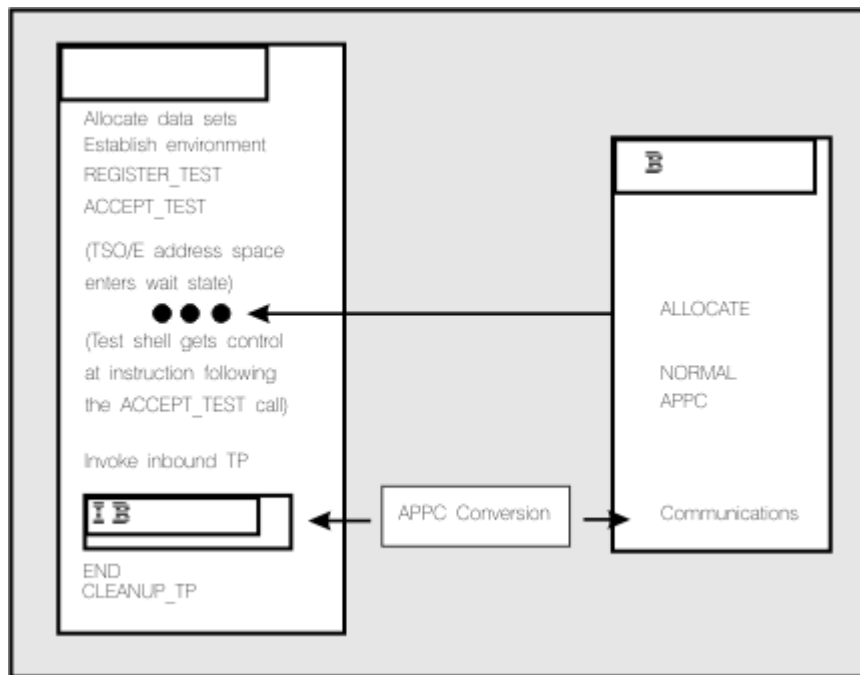


Figure 23. Use of Test Services by a TP Test Shell

Note: If the call to Register_Test fails, APPC resources might be active in the TSO/E address space where the TP is to run. To initialize your environment, you should logoff TSO/E, and then logon again. Do not logon to TSO/E again without first logging off (this will not initialize your environment).

8. Invoke an outbound TP to start a conversation with the inbound TP (the TP to be tested) under APPC/MVS. The outbound TP should specify the following values when calling the Allocate service to allocate the conversation:

- The user ID under which the TP under test (the inbound TP) is running
- A Security_type of security_pgm or security_same.

See Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125 for a complete description of the parameters that are required on an Allocate call from an outbound TP.

9. Run the load module that contains the TP to be tested (you must take this step because the APPC/MVS test services ignore any JCL in the TP profile).
10. Enter calls to the Cleanup_TP and Unregister_Test services from the TP to be tested.
11. When the test is complete, end the TP under test. You can call one of the following Cleanup_TP services to deallocate and clean up any conversations and resources that are left after the test is complete:
 - ATBCUC1, if the test shell is problem state and PSW key 8
 - ATBCTP1, if the test shell is supervisor state or PSW key 0-7.

If you want to repeat a test of a TP, you must call the Register_Test service again.

Using the TSO/E TEST Command to Test an Assembler Language TP

The TSO/E TEST command offers interactive test facilities that allow you to test TPs written in assembler language. TSO/E TEST automatically calls the Register_Test and Accept_Test services, and loads the module for the TP to be tested. When the test ends, TSO/E TEST automatically cleans up the TP to be tested and its conversations if the KEEPTP parameter is **not** specified on the command.

Follow these steps when using TSO/E TEST to test an assembler TP:

1. Logon to TSO/E with a logon procedure that does not allocate APPC/MVS conversations when an APPC/MVS TP is to be tested.

2. Catalog the data set that contains the load module for the TP to be tested.
3. Allocate all required data sets to the TSO/E address space where the TP is to run, including a data set for the TP message log. See the section on logging transaction program processing in [z/OS MVS Planning: APPC/MVS Management](#).
4. Ensure that no active APPC conversations are in progress in the TSO/E address space where the TP is to run.
5. Establish the required environment for the test shell TP, as described in [“Test Shell Characteristics”](#) on page 68.
6. Ensure that the TP under test belongs to an APPC/MVS local LU that allows user-level TP profiles. (Although you are not required to have a user-level TP profile when testing a TP under a TSO/E user ID, the associated LU still must be enabled for user-level TP profiles.) See [“Enabling an LU for User-Level TP Profiles”](#) on page 73.

Figure 24 on page 71 shows a terminal screen that represents the sequence of events for steps 7 through 9, which follow. In the figure, capitalized text indicates a response from the system, and mixed-case text indicates a user command:

```

READY
7 test 'user.appc.load(testtp)' tp('TESTTP') lu('LUA')
      IKJ57522I YOU CAN ALLOCATE THE TP NOW
8 +++ the user starts a program that allocates
  the TP to be tested +++
TEST
9 +++ the user can now test the TP as if it
  is an ordinary program +++
TEST
10 end
READY

```

Figure 24. TSO/E Terminal Screen

7. Enter the TEST command, where "user.appc.load(testtp)" is the load module for the TP to be tested, TESTTP is the TP name under which the load module is to be tested, and LUA is the LU on which the TP is to be tested. If you do not enter the LU keyword, the default value is the system base LU. The system loads and invokes the TP.
8. After the system issues message IKJ57522I, start an outbound TP that allocates an APPC/MVS conversation with the TP to be tested. The outbound TP should specify the following values when calling the Allocate service to allocate the conversation:
 - The user ID under which the TP under test (the inbound TP) is running
 - A Security_type of security_pgm or security_same.

See [Chapter 8, “APPC/MVS TP Conversation Callable Services,”](#) on page 125 for a complete description of the parameters that are required on an Allocate call from an outbound TP. The system displays a TEST mode message.
9. Enter TEST subcommands to test the TP (see [z/OS TSO/E Command Reference](#) for information about TEST subcommands). When the test completes, the system returns another TEST mode message, and waits for the user's next command.
10. Enter the **end** subcommand to end test processing. TEST returns to READY mode.

For complete information about how to use the TSO/E TEST command, see [z/OS TSO/E Command Reference](#) and [z/OS TSO/E Programming Guide](#).

Testing a TP under APPC/MVS Scheduling

Before you test an inbound TP in a live system under APPC/MVS scheduling, ask your APPC administrator to:

- Create a TP profile that restricts the use of the TP under test to your own user ID (called a *user-level TP profile*). See [“Requesting a User-Level or Group-Level TP Profile”](#) on page 72 for an explanation.
- Provide side information for the TP, if necessary. See [“Requesting Side Information”](#) on page 73 for instructions.

After performing these actions, use one of the methods described in [“Methods You Can Use to Create a Test Shell”](#) on page 67 to test a TP under APPC/MVS scheduling.

While the TPs are running, the system writes messages to the job log. When the TPs are finished, check the job log and verify the results of the TP's processing. For information on how to collect problem data from the job log, see [“Collecting Problem Data for Errors that Occur During Testing”](#) on page 73.

Note: If a multi-trans TP remains active indefinitely, the job-log messages for the TP will eventually wrap and overwrite previous messages. However, the messages from the multi-trans shell are not overwritten.

Requesting a User-Level or Group-Level TP Profile

When setting up the APPC/MVS network, your APPC administrator creates a **TP profile** to contain scheduling and security information that might be necessary to run your TP in MVS. The APPC administrator can submit batch jobs to update entries in the TP profile.

A **user-level** TP profile is a TP profile that specifies that only a single user can execute a TP or access the TP's profile. A user-level TP profile prevents other system users from accessing the TP while it is still under test. A **group-level** TP profile is a TP profile that specifies that a specific group of users can run a TP or access the TP's profile.

You might need to have an APPC administrator set up a user-level or group-level TP profile for you. Provide the following information to the APPC administrator when requesting a user-level profile:

- Your user ID, as it is known to your security system (if you are using RACF, this might be your TSO/E user ID).
- The 1- through 64-character TP name that the calling partner passes on its Allocate request to the TP under test.
- An "active status" of YES (to activate the TP).
- The name of a transaction scheduler exit, if your TP is not scheduled by the APPC/MVS transaction scheduler, ASCH.

If your TP uses another scheduler, the rest of the profile contents will vary. If your TP uses the APPC/MVS transaction scheduler, provide the following additional information:

- SYSOUT and account tailoring (whether you want the TP's SYSOUT and account information to be tailored with information from your RACF profile).
- Scheduler class (defined by a system programmer).
- TP schedule type (standard or multi_trans).
- JCL, including a job statement, to schedule and attach the program. Use a unique jobname to help track and debug the TP.
- The name of a data set for the TP message log to record transaction program events. The default name of the TP message log is ‘&SYSUID.&SYSWUID.&TPDATE.&TPTIME.JOBLOG’.
- The KEEP_MESSAGE_LOG parameter value: ALWAYS or ERROR. For testing, use ALWAYS.

Note: The JCL can include conditional statements to print the TP message log on selected error conditions.

- The status of the TP message log (new, old, or mod).
- Any SMS storage, management, or data classes for the TP message log.

When creating a group-level TP profile, the APPC administrator can replace the user ID defined in the user-level profile with a RACF **group-id**, which identifies a group of user IDs. The APPC administrator can enter RACF commands to define the group-id and connect user IDs to the group-id.

To request a group-level TP profile, provide the APPC administrator with the user IDs of all users to be assigned to the group (in addition to the above information). The APPC administrator can then replace the user ID in the TP profile with a RACF group-id.

For more information about the TP profile contents, including examples and JCL restrictions, see [z/OS MVS Planning: APPC/MVS Management](#).

Requesting Access to a User-Level TP Profile

After the APPC administrator creates a user-level TP profile for a TP under test, you should have the security administrator give you access to the profile. If necessary, have the security administrator define the TP profile as a RACF resource, and give your user ID (or the user ID passed on the Allocate request to the TP under test) access to run the TP.

Requesting Side Information

If your TP on MVS uses a symbolic destination name to call a partner TP, your APPC administrator must create an entry in the side information file for the TP. The entry must include:

- The partner TP name (the name of the inbound TP that allocates a conversation with the TP under test)
- The LU and mode names to be associated with the symbolic destination name for the partner TP.

If necessary, have your security administrator give you access to the entry in the side information that contains the listed data.

Enabling an LU for User-Level TP Profiles

When testing a TP either under a TSO/E user ID or under APPC/MVS scheduling, you must ensure that the associated local LU is defined to handle a **user-level TP profile**, which is described in “Requesting a User-Level or Group-Level TP Profile” on page 72. To enable an LU for user-level TP profiles, specify the TPLEVEL(USER) parameter on the LUADD statement in the APPCPMxx parmlib member that defines the local LU.

The following example shows how to create an LUADD statement that enables user-level TP profiles for TPs associated with LU MVSC6LU1:

```
LUADD ACBNAME(MVSC6LU1) /* Add LU to APPC/MVS configuration */
      SCHED(ASCH)       /* Specify that APPC/MVS transaction */
                        /* scheduler is associated with LU */
      TPDATA(APPC.APPCTP) /* The VSAM data set APPC.APPCTP is */
                        /* permanent repository for the TP */
                        /* profiles for this LU. */
      TPLEVEL(USER)      /* Indicates that this LU can process */
                        /* incoming requests at all audience */
                        /* levels, including those that limit */
                        /* use of a TP to individual users. */
```

For more information about how to set processing levels for LUs, see the "Planning Sessions" chapter in [z/OS MVS Planning: APPC/MVS Management](#).

Collecting Problem Data for Errors that Occur During Testing

For errors that occur during testing, some possible problem determination actions include:

- Reading the job log to find system completion (abend) codes.
- Checking the return codes from the APPC services, if they are involved. For example, add messages to write each return code to the terminal or the message-log data set.

- Making sure that the TP handles all possible return codes from each APPC service. For example, if one TP might issue Send_Error, the partner TP must be prepared to handle the Program_error return codes from each APPC service that it calls.

Displaying APPC Activity on MVS

If necessary, a system operator or TSO/E user with operator authority can use the DISPLAY command with the APPC operand to find information about an active conversation. Supply your user ID; for example, the following syntax would display activity for all TPs running under user ID NATHAN:

```
DISPLAY APPC,TP,ALL,USERID=NATHAN
```

The DISPLAY command provides information such as the local and partner TP names, the conversation idle time, the local jobname (ASNAME=xxx), and address space ID (A=xxx). If necessary, an operator can cancel the local TP by jobname and address space ID.

Tracing APPC Conversations

The primary tools for debugging errors in APPC/MVS TPs are described in [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,”](#) on page 77, which includes a section that helps programmers decide which tool to select.

When APPC problems might involve the network or system configuration, system programmers can use APPC component trace to collect diagnostic information. Use the TRACE CT command to start APPC tracing and the IPCS CTRACE subcommand to format the trace data. See [z/OS MVS Diagnosis: Reference](#) for more information.

For conversations that cross the VTAM network, VTAM traces can help diagnose communications problems. See [z/OS Communications Server: SNA Diagnosis Vol 1, Techniques and Procedures](#) for more information.

The MVS and VTAM trace facilities can return detailed information about a conversation, including data that passed between the TPs. For example, the FMH-5 (Functional Management Header-5) contains the input for an inbound allocate request; APPC passes that input in the FMH-5 across the network to the partner LU. To understand the contents of an FMH-5, see *SNA Formats*.

Putting a Tested TP into Production

The way you make a tested TP available for general use depends on the policy of your installation. Your APPC administrator can do the following to enable other users to run the TP:

1. Copy the user-level TP profile that the APPC administrator created for the test shell into a profile data set associated with another LU (ideally an LU dedicated to production-ready TPs).
2. In the profile, replace the parameter that specifies your user ID with one of the following:
 - The SYSTEM parameter (to make the profile available to the LU).
 - A RACF "group ID" to create a **group-level** TP profile (to make the profile available to a group of users). If your security system is not RACF, specify an identifier that defines a group of users to the security system.

See [“Requesting a User-Level or Group-Level TP Profile”](#) on page 72 for information.

3. Edit the profile's JCL to set the correct environment for the new users (if necessary). For example, change the KEEP_MESSAGE_LOG parameter value from ALWAYS to ERROR.

Your security administrator might also need to give the appropriate users access to the TP profile and any other TPs that the TP allocates conversations with, including any side information used on the allocate requests.

Replacing an Active TP

At some point you might need to replace an inbound TP that is currently active on your system. To avoid having old and new versions active at the same time, have an APPC administrator:

- Set the current TP profile "inactive" to prevent new inbound requests
- Allow the active transaction initiator to process any queued requests
- Modify the TP profile for the new version if necessary and reactivate the TP profile, or delete the TP profile and add a new one.

Chapter 6. Diagnosing Problems with APPC/MVS TPs

Several tools are available to help you diagnose problems related to APPC/MVS transaction programs (TPs). Like the great detectives of fiction, each tool has its own specialty. Just as Sir Arthur Conan Doyle's Sherlock Holmes collects details from the scene of the crime, so APPC component trace collects information about the environment in which APPC/MVS TPs run. In contrast, Agatha Christie's Hercule Poirot relies on conversing with each suspect to uncover facts, as the application programming interface (API) trace facility does by collecting details about conversation calls that an APPC/MVS TP issues.

While you might hire your favorite fictional detective to solve any mystery, you achieve better results if you select a diagnostic tool based on the problem symptoms, rather than using the same tool for all errors. In some cases, you might need a combination of tools to ensure that you are collecting the most pertinent diagnostic information. For example, if you are testing an APPC/MVS TP that converses with a TP on OS/2, you might need to use both the API trace facility and OS/2's tracing facility to debug problems with the conversation flow.

You also might use one or more tools even when an error has not occurred. Again, in a test environment, you could use the TP message log and the API trace facility to collect data each time the TP runs. If errors occur during the TP's processing, you already have the diagnostic data you need to debug the errors; if the TP successfully finishes its processing, you already have the data you need to verify its success.

In most cases, the API tracing facility can provide you with the diagnostic data you need. Additionally, it has the advantage of being relatively easy to start and stop. However, because it might not be the most appropriate tool in all cases, use [Table 9 on page 77](#) to help you select which tool is best suited to the error conditions you observe, or might expect to encounter, for an APPC/MVS TP.

Table 9. Selecting a Diagnostic Tool

When the symptoms are:	The usual suspects are:	The detective of choice is:
Non-zero return code from an APPC/MVS or CPI-C service call	<ul style="list-style-type: none"> Unknown parameters passed on the service call Incorrect parameter values passed on the call Incorrect conversation state for issuing the call to the service Conditions that prevent the successful completion of the service call 	API trace facility, or Error_Extract service
Return code for product-specific error	<ul style="list-style-type: none"> Incorrect version of service call for the system on which the TP is running APPC/MVS is not active, or has encountered an internal error Various environmental errors, which can include security checking, caller authorization, APPC/MVS configuration errors, and so on 	API trace facility, Error_Extract service, or the symptom record for the error

Table 9. Selecting a Diagnostic Tool (continued)

When the symptoms are:	The usual suspects are:	The detective of choice is:
<ul style="list-style-type: none"> • Unexpected results or non-zero return codes for TP processing • Unexpected DISPLAY command results about the status of the TP or its partners • TP or its partner seems to be hanging • Abnormal end of the TP 	<ul style="list-style-type: none"> • Incorrect sequence of calls between partner TPs • Incorrect TP design • Incorrect parameter syntax • Incorrect conversation state 	API trace facility
<ul style="list-style-type: none"> • Unexpected results or non-zero return codes for TP processing • Problems related to the allocation of resources on behalf of the TP • Abnormal end of the TP 	<ul style="list-style-type: none"> • Non-zero return codes from MVS macros or callable services • Incorrect TP profile or side information contents 	TP message log
Error messages or unexpected DISPLAY command results about the status of the APPC/MVS configuration	<ul style="list-style-type: none"> • Status of the LU, scheduler, or server for the TP • System errors or environmental conditions that affect the processing of APPC/MVS 	IPCS APPCDATA or ASCHDATA subcommands, or APPC/MVS component trace

As application programmers designing and coding APPC/MVS TPs, your primary diagnostic tools are the API trace facility, Error_Extract, and the TP message log, which are described in this chapter, along with descriptions of symptom record contents for product-specific errors. The API trace facility and TP message logs require some installation set-up, which is usually performed by system programmers; *z/OS MVS Planning: APPC/MVS Management* contains this set-up information. Information about other diagnostic tools appears elsewhere:

For this tool:	Information appears in:
APPCDATA and ASCHDATA subcommands	<i>z/OS MVS Diagnosis: Reference</i>
APPC/MVS component trace	<i>z/OS MVS Diagnosis: Tools and Service Aids</i>

Comparing the Detectives: Error_Extract, API Trace, and the TP Message Log

In the distributed processing environment, debugging programming errors is much like solving a complicated murder mystery; your goal is to find out what killed your TP. Fortunately, APPC/MVS provides the following diagnostic tools to help you solve that mystery:

- The API trace facility,
- The Error_Extract callable service, and
- The TP message log.

These tools differ in several ways that can help you determine which tool, or combination of tools, is best suited to detect the cause of the error. The following topics describe these differences in general terms; for detailed information about each tool, see:

- [“Diagnosing TP Conversation Errors with the API Trace Facility” on page 82](#)

- “Overview of Error_Extract Service” on page 108
- The section on logging TP processing in *z/OS MVS Planning: APPC/MVS Management*.

Clues: What Information They Collect

Sherlock Holmes always concentrates fiercely on physical evidence; he never visits the scene of the crime without his magnifying glass. Hercule Poirot, in marked contrast, disdains crawling in the dirt, on hands and knees; instead, he prefers to mingle with the suspects, collecting significant words and phrases that explain people's behavior and psychology.

As with Holmes and Poirot, each APPC/MVS diagnostic tool collects different types of clues. Error_Extract provides detailed information about the error that occurred on a single call to an APPC/MVS or CPI-C service, whereas API trace provides the details about each service call made by a particular TP, regardless of whether an error occurred. When an error does occur on a specific call, the API trace data for that call automatically includes the information that Error_Extract would return for that service. Because API trace provides the same, and more, information as Error_Extract, API trace is the preferred detective for most situations.

API trace and Error_Extract provide diagnostic information for most, but not all, APPC/MVS and CPI-C callable services. See [Table 10 on page 81](#) for a list of the services that are supported by either API trace, Error_Extract, or both.

In contrast, the TP message log provides information through messages generated while a TP runs, such as system messages about resources allocated by the TP. The message log could contain messages about a conversation call, but only if the application programmer deliberately codes the TP to issue write-to-programmer messages to record its progress. Although useful, these messages cannot contain the detailed information that APPC/MVS provides through the API trace facility.

Modus Operandi: How They Interrogate Suspects

Because Holmes relies on physical evidence, his contact with suspects is usually brief and impersonal; he works almost in isolation, with little contact with others. Poirot, on the other hand, mingles with the suspects on a daily basis, sometimes becoming a temporary part of the household until the mystery is solved.

Similarly, the APPC/MVS diagnostic tools either work independently of TP processing, or can be integrated into the TP's design. You might use the TP message log to collect write-to-programmer messages only during testing; collecting such footprints is usually a temporary task, rather than being a permanent, integral part of the TP's design. Calls to Error_Extract, however, can be permanently integrated into the design of any APPC/MVS TP. Whenever the TP's processing is based on the results of a call to an APPC/MVS or CPI-C service, the TP can call Error_Extract immediately after detecting a non-zero value for the service return code. With the additional diagnostic information that Error_Extract returns, the TP can more efficiently process unsuccessful calls, even if it does nothing more than pass that information to its caller or an end user.

A call to Error_Extract does not have to be a permanent part of the TP's design, but inserting calls for testing and then removing them for production would be relatively inefficient and time-consuming. Instead, you could simply turn on the API trace facility during testing, and turn it off during production. Because the API trace facility is provided through a REXX exec instead of a callable service, you can control tracing activity in several ways, without invoking the exec directly in the TP's code. This flexibility, along with the amount of trace data collected, makes the API trace facility easier and more efficient to use.

Fees: How to Reduce the Cost of the Investigation

Only the rich can afford to hire Sherlock Holmes or Hercule Poirot, but you don't necessarily have to pay a high price for using the APPC/MVS diagnostic tools. Each tool offers options that can help you reduce the cost of diagnosing errors.

Because API tracing is not necessarily controlled only through calls in a particular TP's code, tracing activity can encompass many TPs, many conversations, and many users. Through only one invocation, you can start API tracing activity in a test or production environment, and later stop the tracing activity at any time, just as quickly and easily. This flexibility has a price: the more API traces you activate, the greater the possible impact on the performance of APPC/MVS work. In a test environment, you might be able to tolerate possible performance degradation; in a production environment, you should consider restricting trace activity to reduce or eliminate any performance impact.

Although TP message logging is directly tied to a TP's processing, it might have similar global effects on APPC/MVS work. Depending on the way your installation has set up TP message logs, storage constraints or I/O contention might result. To reduce or eliminate these possible effects, define a TP message log that is cumulative (or reuse an existing data set), to be used only when an error occurs.

Because it is a callable service, Error_Extract is directly tied to a TP's processing, but has little or no effect on that TP's performance. However, for other reasons, such as the effort of integrating Error_Extract calls into a TP's design, consider using Error_Extract only when your TP's processing depends on the return code from an APPC/MVS or CPI-C service, or immediately after a failure in an Allocate call. (Calling Error_Extract immediately after an Allocate call is the only method of obtaining diagnostic information when the call fails because of problems with the TP or LU.) In most other situations, the API trace facility is much easier to use to collect diagnostic data.

The Initial Consultation: Building Your Crime Lab

Hiring either Sherlock Holmes or Hercule Poirot usually requires some effort. If Holmes' mood is unfavorable, only the most persistent client can engage his services. For Poirot, the nature of the mystery must be sufficiently intriguing to entice him to work. Depending on your installation, some APPC/MVS tools require initial set-up before programmers can use them, just as prospective clients must provide background information for Holmes and Poirot.

Both the API trace facility and TP message log require some installation set-up, which is usually performed by a system programmer. Both require data sets that they use to contain trace data or TP messages; in some cases, these data sets must be pre-allocated. In addition to pre-allocating data sets, the system programmer might have to perform the following tasks:

- Define security profiles for the APPC/MVS resources related to tracing activity.
- Provide TP message log information in ASCHPMxx parmlib members and TP profile files.

The Error_Extract service requires no installation set-up.

The All-Star Collaboration: A Team Approach

Fictional detectives do not often team up to solve a particularly complex mystery; their egos are usually too great to allow such collaboration. The APPC/MVS detectives are different; they are quite compatible. To muster these detectives into an efficient mystery-solving team, consider using the following strategy:

- Using Error_Extract calls:
 - When a TP's processing varies depending on the return code from an APPC/MVS or CPI-C service call.
 - After any Allocate calls to establish a conversation with a TP. Probably the most common error is making sure your program correctly identifies the TP so that APPC/MVS can find and run it. If the TP does not get control, Error_Extract information can help you pinpoint errors in configuration.
- For a test system:
 - Setting up message logging so that a TP message log is produced for each instance of a TP, even when errors do not occur. Once the TP's execution environment appears to be relatively stable (for example, resources are being allocated correctly), reduce message logging to error situations.
 - Setting up and starting an API trace for the TP, limiting tracing activity to a few users. Once the conversation flow between the TP and its partners appears to be correct, and individual service calls are handled correctly, tracing activity can be expanded to encompass more users. More closely

approximating a production environment might make interpreting trace data more difficult, but might allow you to detect more potential problems.

- For a production system:
 - Reducing message logging to a minimum. Using cumulative logs, and writing messages only for error situations, can significantly reduce storage constraints and I/O, and might prevent you from having to re-create problems to obtain diagnostic data.
 - Using the API trace facility only when errors occur. As with message logging, some performance degradation might occur during tracing, but you can limit its use to specific TPs, conversations, or users to reduce the impact to your production system.

Calls that Error_Extract or API Trace Support

Neither Holmes nor Poirot accepts any cases involving divorce. Similarly, the Error_Extract service and API trace facility do not collect information for certain conversation calls. Generally speaking, the unsupported calls either do not directly affect a TP's conversation, or do not get used frequently.

Table 10 on page 81 lists the conversation services that are supported by either Error_Extract, API trace, or both. Callable services are listed by descriptive name; for example, "Allocate" represents both the CPI-C verb CMALLC, and the APPC/MVS verbs ATBALLC and ATBALC2. All versions of each service are supported, so existing TPs do not have to be changed to invoke the most recent version. The API trace facility also supports some APPC/MVS allocate queue services; if necessary, see [z/OS MVS Programming: Writing Servers for APPC/MVS](#) for more information.

Error_Extract and API trace return information about a conversation call only when:

- APPC/MVS is active
- A valid conversation ID is passed on those verbs that accept a conversation ID as a supplied parameter

For API trace only, additional requirements, described in [“How APPC/MVS Handles an ATBTRACE START Request”](#) on page 86, determine whether information is collected for the conversation in which the call is issued.

Table 10. Calls that Error_Extract or API Trace Support			
Callable Service Name	APPC/MVS or CPI-C call	Supported by:	
		Error_Extract	API trace facility
Accept_Conversation	CPI-C	Yes	Yes
Allocate	Both	Yes	Yes
Cleanup_TP	APPC/MVS	No	Yes
Confirm	Both	Yes	Yes
Confirmed	Both	Yes	Yes
Deallocate	Both	Yes	Yes
Extract_Conversation_State	CPI-C	Yes	Yes
Extract_Conversation_Type	CPI-C	Yes	Yes
Extract_Mode_Name	CPI-C	Yes	Yes
Extract_Partner_LU_Name	CPI-C	Yes	Yes
Extract_Sync_Level	CPI-C	Yes	Yes
Flush	Both	Yes	Yes
Get_Attributes	APPC/MVS	Yes	Yes

<i>Table 10. Calls that Error_Extract or API Trace Support (continued)</i>			
Callable Service Name	APPC/MVS or CPI-C call	Supported by:	
		Error_Extract	API trace facility
Get_Conversation	APPC/MVS	Yes	Yes
Get_TP_Properties	APPC/MVS	No	Yes
Get_Type	APPC/MVS	Yes	Yes
Initialize_Conversation	CPI-C	Yes	Yes
Post_on_Receive	APPC/MVS	Yes	Yes
Prepare_to_Receive	Both	Yes	Yes
Receive	CPI-C	Yes	Yes
Receive_Immediate	APPC/MVS	Yes	Yes
Receive_and_Wait	APPC/MVS	Yes	Yes
Reject_Conversation	APPC/MVS	No	Yes
Request_to_Send	Both	Yes	Yes
Send_Data	Both	Yes	Yes
Send_Error	Both	Yes	Yes
Set_Conversation_Type	CPI-C	Yes	Yes
Set_Deallocate_Type	CPI-C	Yes	Yes
Set_Error_Direction	CPI-C	Yes	Yes
Set_Fill	CPI-C	Yes	Yes
Set_Log_Data	CPI-C	Yes	Yes
Set_Mode_Name	CPI-C	Yes	Yes
Set_Partner_LU_Name	CPI-C	Yes	Yes
Set_Prepare_to_Receive_Type	CPI-C	Yes	Yes
Set_Receive_Type	CPI-C	Yes	Yes
Set_Return_Control	CPI-C	Yes	Yes
Set_Send_Type	CPI-C	Yes	Yes
Set_Sync_Level	CPI-C	Yes	Yes
Set_Syncpt_Options	APPC/MVS	No	Yes
Set_TP_Name	CPI-C	Yes	Yes
Test_Request_to_Send_Received	CPI-C	Yes	Yes

Diagnosing TP Conversation Errors with the API Trace Facility

Using the application programming interface (API) trace facility, you can collect data about APPC/MVS and CPI-C calls that an APPC/MVS TP issues. With this data, you 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 reduces the amount of time and effort required to develop and service APPC applications.

Through the API trace facility, you can collect the following data for a particular TP running on a particular LU:

- Parameters and values specified on calls issued for APPC/MVS and CPI-C services, and values provided on return from those calls.
- The same diagnostic information that the APPC/MVS Error_Extract service provides for calls that return non-zero return codes.
- Parameters and values specified on START and STOP requests for the API trace facility.
- The contents of FMH-5 or FMH-7 records exchanged between conversing TPs.

This API trace data is stored in a pre-allocated data set that you specify when starting the trace. You can view the data only after all tracing activity for the data set has stopped.

To control API tracing activity, use the ATBTRACE REXX exec to start or stop tracing, or to list the status of active API traces. Through the ATBTRACE exec, you specify:

- The type of trace request: START, STOP or LIST
- The LU and TP combination that identifies the conversation
- The pre-allocated data set for storing trace data.

You can invoke the ATBTRACE exec from TSO/E, through JCL for a TP or batch job, or from a high-level language program.

Depending on the location of the partners, you might have to start tracing activity on more than one z/OS system, or use the API trace facility together with tracing facilities provided for other operating systems, such as OS/2.

The following sections explain how to use the API trace facility:

- [“Setting Up API Trace Data Sets” on page 83](#) describes 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.
- [“Starting API Tracing Activity” on page 86](#) explains how to select values for the ATBTRACE START request, to ensure that APPC/MVS collects the trace data you need.
- [“Using the ATBTRACE REXX Exec” on page 93](#) explains:
 - The methods of invoking the ATBTRACE REXX exec, including programming considerations and output
 - The syntax diagrams and parameter descriptions for each type of ATBTRACE request.
- [“Interpreting API Trace Data” on page 103](#) contains general descriptions and examples of trace data set entries, and provides suggestions for sorting and reading the entries.

Setting Up API Trace Data Sets

Before you start using the API trace facility, you must have at least one pre-allocated, sequential data set to contain the trace data. APPC/MVS limits the number of API trace data sets per z/OS system to fifty. This limit is imposed to control the possible impact on performance; collecting and writing trace data are resource-intensive activities that might affect not only APPC/MVS work, but also other system work. On a test system, performance might not be a major concern; on a production system, you may use API tracing options that further limit tracing activity to an acceptable level, with little or no impact on performance.

In any case, because of the limited number of trace data sets, your installation might have to consider various approaches to setting up the API trace data sets, to ensure greatest efficiency and ease of use. Installation set-up affects what you need to know before using the API trace facility:

- Security requirements for programmers who use the data sets
- 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
- Suggestions that might increase efficient use of the trace data sets.

Getting Access to APPC/MVS Resources for Tracing Activity

What you need to do to access APPC/MVS resources depends on a number of factors:

- If your installation uses RACF or an equivalent security product to protect APPC/MVS resources, you might need access to one or more of the following:
 - The API trace facility resource
 - The API trace data sets
 - The APPC/MVS LUs and TPs to be traced.

z/OS MVS Planning: APPC/MVS Management contains more details about these security requirements.

- If your installation has many application programmers, it might choose to have a system programmer set up the all data sets for API tracing activity. If so, you need to have UPDATE access to the data sets before you can start tracing. Otherwise, you may create your own trace data sets; in this case, no special authority is required.
- Again, for a large population of application programmers, your installation might have to require you to share the same trace data set with other programmers. If you and others share the same data set, and want to trace TPs concurrently, be aware of the following:
 - Although APPC/MVS allows multiple ATBTRACE START requests while the data set is open and traces are active, all requests have to be issued from the same user ID from which the first request was issued. Otherwise, the data set is considered in use, and the START request fails.
 - Depending on your installation's security policies, more than one user may issue a STOP request for the data set. Although this flexibility can prove useful in critical situations, it might pose problems if more than one programmer has requested concurrent tracing activity for the data set. A STOP request stops all active traces for the data set, so issuing a STOP request might adversely affect someone else's work.

Determining Data Set Characteristics

APPC/MVS requires API trace data sets to be pre-allocated and sequential, with a block size of 4096 or larger. (The block size requirement applies only if you specify the BLKSIZE operand.) If you already have data sets that meet those requirements, you might be able to use them without having to allocate others. Whether you are using existing data sets, or setting up new data sets, keep the following points in mind:

- APPC/MVS uses only primary space for trace data sets. If tracing activity is not stopped before APPC/MVS reaches the end of the primary space, and the data set resides on DASD, APPC/MVS begins wrapping the trace data by returning to the beginning of the data set, and overwriting older trace entries with more recent entries. APPC/MVS does not indicate how many times wrapping occurs.
- APPC/MVS uses variable blocked record format, regardless of the format specified during allocation, but does use the block size, if you specify the BLKSIZE operand. Depending on the specified block size, and I/O contention, APPC/MVS might collect trace data faster than it can write that data to the data set. When this backlog of collected data becomes too large, APPC/MVS temporarily stops collecting trace data until it can finish writing the backlog to the data set. APPC/MVS indicates any suspension of tracing activity through an entry in the trace data.

Both of these situations result in the loss of trace data, which might hamper your ability to diagnose problems with the TPs being traced. They also might make reading the trace data more difficult. Fortunately, several techniques can help you reduce the possibility of losing trace data in either situation, and reduce the relative complexity of interpreting trace data.

Avoiding Loss of Data through Wrapping

To avoid losing trace data through wrapping, you need to make sure the primary space allocated for the data set is sufficient to contain all the trace entries. Determining the amount of primary space required might be a trial-and-error process, depending on how accurately you can estimate:

- The number of LU/TP (and possibly user ID) combinations that will be traced into a particular data set. APPC/MVS does not limit the number of unique LU/TP combinations specified for tracing into a particular data set, but the more you specify, the greater the chance that wrapping will occur.
- The number of calls your TP issues for APPC/MVS or CPI-C services. The more calls your TP issues (especially asynchronous calls), the more trace data will be generated.
- How many instances of the TP run. If you have a TP that might run many times while tracing is on, the cumulative effect of its calls might easily fill the data set.

You might have to run a TP and review the collected trace data several times, and adjust the primary space accordingly, before determining an adequate amount of primary space. (See [“Reading Trace Entries When Wrapping Occurred”](#) on page 106 if you need to determine whether wrapping has occurred.)

An alternative, which might be easier than determining the primary space for DASD, is to use a tape data set for the trace data. With tape, APPC/MVS can write trace entries into a total of five volumes. If you need more space, you can allocate the tape with a volume count, which increases the total to 255 volumes. The disadvantage to this approach is reduced usability: the trace data on tape has to be moved or copied over to DASD before you can view it.

Avoiding Loss of Data through Suspension of Tracing Activity

To avoid losing trace data because APPC/MVS has to suspend tracing activity, you may try any combination of the following tactics:

- Allow APPC/MVS to determine the optimal block size for the device on which the data set is allocated. To do so, allocate the data set without specifying the BLKSIZE operand.
- Make sure the API trace data sets reside on volumes that are not used heavily by other subsystems or applications. Separating trace data sets from other types of data sets might help reduce contention or delays because of frequent use and serialization.
- Filter ATBTRACE START requests as much as possible. For example, instead of collecting trace data for all instances of a TP, restrict tracing activity to only the conversations allocated for that TP with particular user IDs. Doing so can reduce the amount of trace data that is generated for an LU/TP combination.

Some Suggestions for Data Set Setup...

The following suggestions might help you determine the most efficient, easiest ways to approach API tracing. Some might not be feasible for your installation, but experiment if you can, especially in a testing environment. What you learn under those conditions might easily help you use API tracing most efficiently, when efficiency is most important — during emergency situations in a production environment.

- Do not use a specific block size for trace data sets; allow APPC/MVS to determine the optimal size, to reduce or eliminate the need to suspend tracing activity.
- Overestimate the primary space you need for the trace data set, or use a tape data set and later switch to DASD, especially when first testing a TP. Doing so reduces the chances of wrapping trace entries, and possibly reduces the length of the trial-and-error period to determine the size of an average trace data set for tracing your installation's set of TPs.
- Limit the number of unique LU/TP combinations being traced into a particular data set. For example, use the optional USERID parameter to restrict tracing activity to only specific users. Such limits can prevent:
 - Loss of trace data through wrapping or suspension of tracing activity
 - Complexity of interpreting trace data
 - Adverse effects on performance.

- Try to limit the number of programmers sharing a data set for concurrent traces. This tactic is another way to limit the number of unique LU/TP combinations for a particular data set, so it offers the benefits of that limitation, as well as improving productivity:
 - You can view trace data as soon as you stop the tracing activity for the data set; you do not have to worry about having someone prematurely stop your trace, or about doing the same to another programmer.
 - You do not have to separate your trace data from someone else's, as well as sorting the trace data for separate conversations.

Starting API Tracing Activity

Once you have trace data sets available for use, you can invoke the ATBTRACE REXX exec to start tracing activity for a particular TP. To collect all the pertinent trace data for this TP, you have to know more than just the TP name. Your installation's APPC configuration, and your TP's partners and processing, determine what values you supply on the ATBTRACE exec, and how many times, and where, you might have to issue ATBTRACE START requests. At a minimum, you must supply an LU name and TP name to uniquely identify the conversations to be traced. This LU/TP combination is required because APPC/MVS allows multiple TP profile files, each of which might define a TP with the same name, for a particular z/OS system.

Before you begin issuing ATBTRACE START requests to activate tracing, you need to answer the following questions to decide what values to supply on those START requests:

- Does your TP initiate conversations with partners, or do the partners initiate the conversations?
- Through which APPC/MVS LUs do inbound Allocate requests arrive for the TP? How are the LUs configured; for example, are they members of a VTAM generic resource group?
- Where do the TP's partners reside: On the same z/OS system? On a different z/OS system? On an operating system other than MVS?

Once you have answered these questions, continue to the next topics, which cover:

- How APPC/MVS handles an ATBTRACE START request, which will help you understand how many START requests you need to issue, and with what values
- Suggestions that might increase programmer productivity and efficient use of the API trace facility.

How APPC/MVS Handles an ATBTRACE START Request

Suppose your installation has the following configuration, which is illustrated in [Figure 25 on page 87](#):

- A sysplex of z/OS systems, with APPC/MVS running on each (for simplicity, only two are shown)
- A connected network of OS/2 workstations (for simplicity, only two are shown)
- One set of TP profile files shared by both z/OS systems
- One side information file shared by both z/OS systems
- One set of APPC/MVS LUs in a VTAM generic resource group named MVSLU.

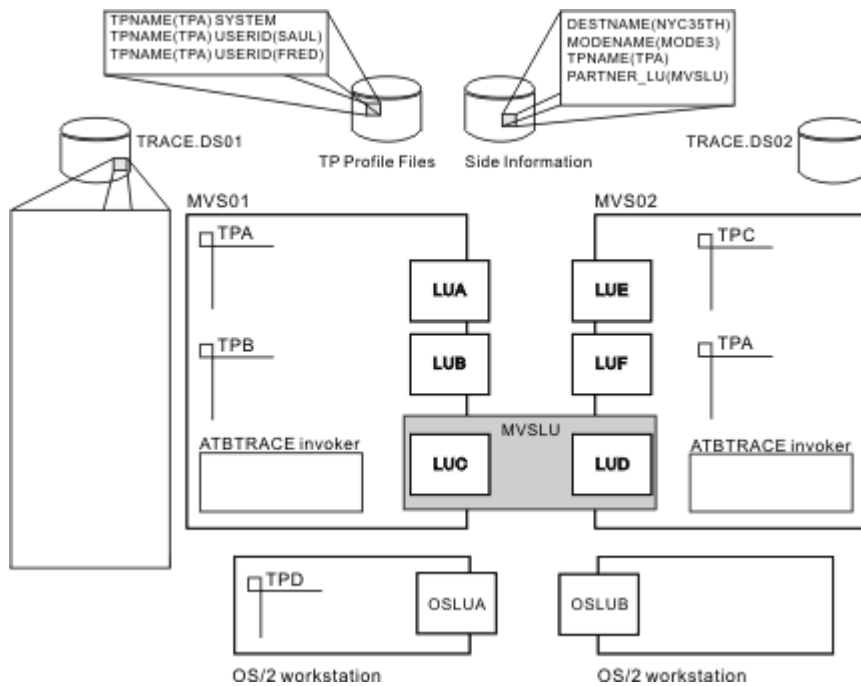


Figure 25. Sample Installation Configuration

Given that configuration, suppose you issue the following ATBTRACE START request on system MVS01, to collect trace data for TPA's conversations:

```
ATBTRACE START DSN(TRACE.DS01) LU(LUA) TP(TPA)
```

Once it finishes processing the START request, APPC/MVS on the MVS01 system compares these LU name and TP name values with those specified (explicitly or implicitly) on all inbound and outbound requests to establish a conversation. If the values match, APPC/MVS collects the trace data for:

- All supported APPC/MVS or CPI-C calls that TPA issues.
- All supported APPC/MVS or CPI-C calls that TPA's partner TP issues for its conversation with TPA, only if the following are true:
 - The partner issued the Allocate call to establish a conversation with TPA.
 - The partner itself did not receive control because of an inbound Allocate request.
 - The partner resides on the same z/OS system as TPA.

If one of those conditions is false, APPC/MVS provides trace data for only the calls issued by TPA.

Read through the following scenarios to understand the implications of APPC/MVS processing for START requests.

Timing: When Tracing Starts

In Figure 26 on page 88, if TPB allocates a conversation with its partner TPA (1a) before you issue the START request (2), APPC/MVS does not provide any trace data for that conversation, even though the LU and TP values match. APPC/MVS provides trace data only for conversations that are established after it finishes processing a START request.

Generally speaking, a conversation is established when a program issues an Allocate call or, in the case of APPC/MVS servers, a Register_for_Allocates call. When the ATBTRACE START request is issued before the conversation is established, trace data begins with a trace entry for the Allocate or Register_for_Allocates call.

For APPC/MVS TPs that issue CPI-C verbs, however, a conversation is established when either:

- The TP issues CMINIT, specifying a valid symbolic destination name. In this case, trace data begins with a trace entry for the CMINIT call.
- The TP issues the last CPI-C call to set either the partner LU name (CMSPLN) or TP name (CMSTPN). For example, suppose the TP issues the following calls in sequence:
 1. CMINIT, without specifying a symbolic destination name
 2. CMSPLN, to set the partner LU name
 3. CMSTPN, to set the TP name.

In this case, trace activity begins with the CMSTPN call.

In Figure 26 on page 88, because TPD does not reside on MVS01, the only trace entry for TPD's side of the conversation is an FMH-5 received at LUA from OSLUA (3a). If TPD did reside on MVS01, trace data would include an ATB60055I message for the CMSTPN call, and messages for subsequent calls by TPD.

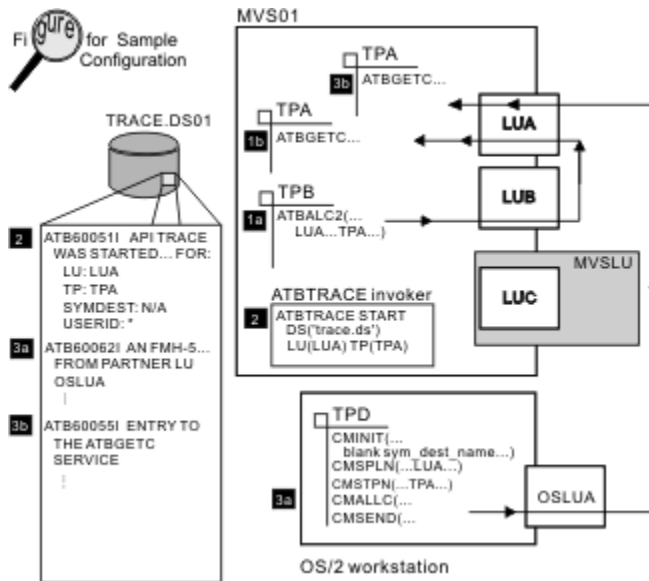


Figure 26. Timing the Start of API Tracing Activity

APPC/MVS does not start tracing activity at all for the LU/TP combination, if the call to establish the conversation contains partner LU name or TP name values that do not match the LU/TP combination specified through the ATBTRACE START request.

Using VTAM Generic Resource Names

If a TP issues an Allocate call specifying the VTAM generic resource name MVSLU, that call can be directed to any of the LUs in the generic resource group. Because you cannot always predict which LU will handle the conversation, you run the risk of not tracing the conversation unless you:

- Specify the generic resource name MVSLU on the ATBTRACE START request, and
- Issue the START request on each system on which an MVSLU group member resides.

For example, suppose you entered a START request specifying TPA and LUC on MVS01, but did not enter the same START request on MVS02. Given the configuration in Figure 27 on page 89, the Allocate request from TPC is directed to LUD, so you would not get any trace data for TPA's calls for its conversation with TPC.

Figure 27 on page 89 shows sample ATBTRACE START requests that IBM recommends for TPs running on MVS LUs in a generic resource group. If your installation uses a VTAM generic resource name for all the LUs on which TPA might run, and you want to collect trace data for all of TPA's conversations regardless of the LU selected to handle the inbound Allocate call, do the following:

- Specify the VTAM generic resource name for the LU keyword value on the START request.
- If the LUs in the generic resource group reside on more than one MVS system:

- Issue the START request on each z/OS system.
- For each START request, specify a different trace data set, because each data set can be in use by only one system at a time.

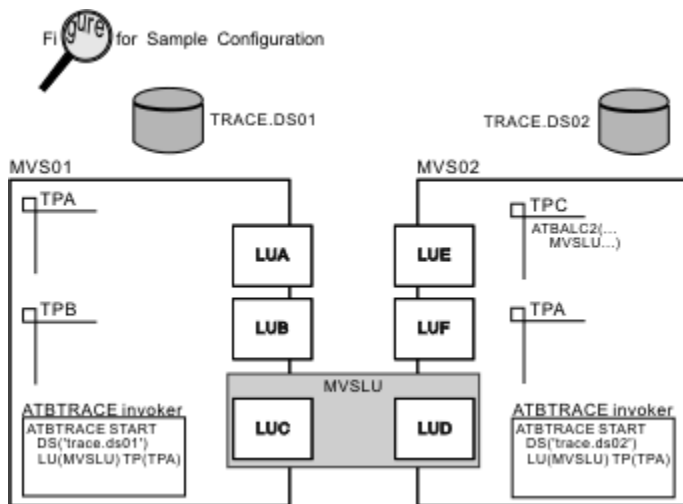


Figure 27. Using VTAM Generic Resource Names

z/OS MVS Planning: APPC/MVS Management contains possible security requirements and general information about VTAM generic resource support for APPC/MVS LUs.

Using Symbolic Destination Names

In Figure 28 on page 89, if TPC issues an Allocate call specifying a symbolic destination name, NYC35TH, that resolves to LUA and TPA, APPC/MVS provides trace data for that conversation. Even though you specified the LU and TP keywords, instead of the SYMDEST keyword, on the START request, APPC/MVS provides trace data for the conversation because the LU and TP values match. If you know that TPA's partners allocate their conversations using a symbolic destination name, however, IBM recommends that you use the SYMDEST keyword and value on the ATBTRACE START request.

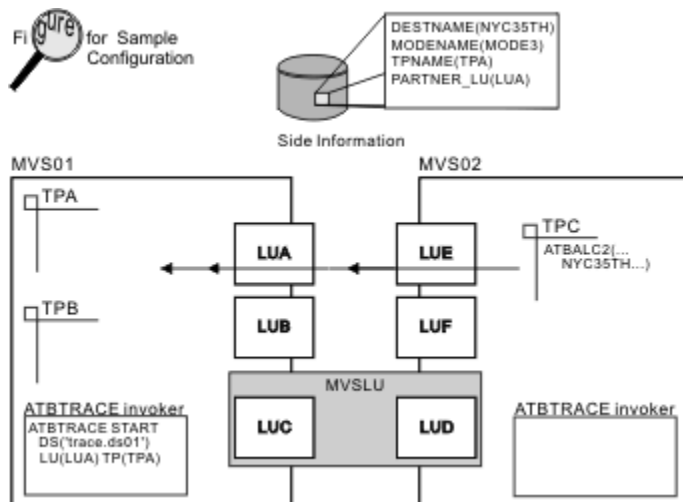


Figure 28. Using Symbolic Destination Names

Collecting Trace Data for Concurrent Conversations

If you want to collect trace data for multiple outbound conversations allocated by a single program, you might be able to do so with only one ATBTRACE START request, even if the LU/TP pairs for the conversations do not match the pair specified on that START request. APPC/MVS automatically collects trace data for a program's additional outbound conversations, as long as both of the following are true:

1. An ATBTRACE START request, specifying an LU/TP pair for which the program allocates a conversation, was issued on the system on which the program runs
2. APPC/MVS has already started collecting trace data for the LU/TP pair specified on the START request (that is, the program allocates the conversation with the LU/TP pair), before the program allocates the additional outbound conversations.

For example, in [Figure 29](#) on [page 90](#), suppose that you have already entered a START request for LUA and TPA (1). Later, TPB allocates a conversation with TPA, and tracing begins (2). While still conversing with TPA, TPB allocates a conversation with TPC (3).

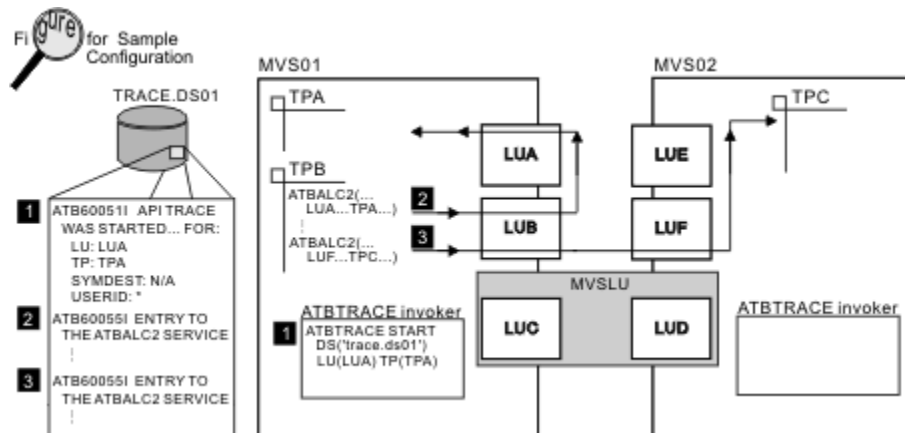


Figure 29. Collecting API Trace Data for Concurrent Conversations

APPC/MVS collects trace data for TPB's calls for its conversation with TPA, because of the START request already entered; APPC/MVS automatically includes TPB's calls for its conversation with TPC, because tracing has already begun for the LUA/TPA pair.

If, however, TPB allocated the conversation with LUF/TPC before the conversation with LUA/TPA, APPC/MVS does not trace the calls issued for the LUF/TPC conversation. Unless you can guarantee that the program first allocates the conversation for the LU/TP pair specified on the START request, you need to enter additional START requests, such as:

```
ATBTRACE START DSN(TRACE.DS01) LU(LUF) TP(TPC)
```

Note that, in this case, you can use the same trace data set for a START request with a different LU/TP combination. This capability allows you to collect, in one place, the trace data for an application program that holds concurrent conversations with different TPs running on different LUs.

If you want to trace the calls made on both sides of a conversation that spans z/OS systems, such as the conversation between TPB and TPC shown in [Figure 30](#) on [page 91](#), enter:

- A START request for the LU/TP combination on one system, specifying one trace data set; for example, LUF/TPC (1a).
- A START request for the LU/TP combination on the other system, specifying a different trace data set, because a trace data set may be in use by only one system at a time; for example, LUF/TPC (1b).

When the conversation between TPB and TPC takes place:

- TRACE.DS01 will contain entries for the calls that TPB issues, and FMH-5 entries for calls received from TPC.
- TRACE.DS02 will contain entries for the calls that TPC issues, and FMH-5 entries for calls received from TPB.

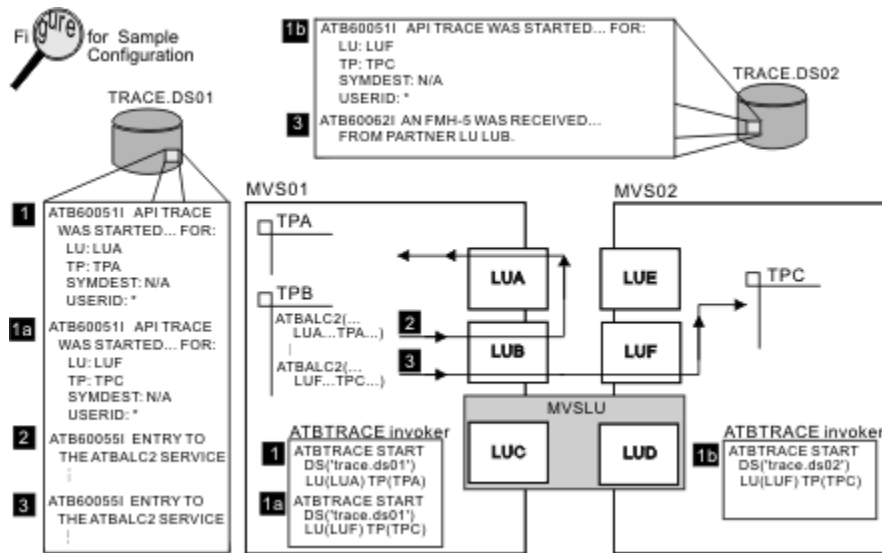


Figure 30. Collecting API Trace Data for Concurrent Conversations

Filtering Trace Data by User ID

By default, an ATBTRACE START request specifies USERID(*), which results in trace data for all conversations for the LU and TP combination, including conversations allocated without a user ID. To filter the amount of trace data, you may specify either the USERID or SECNONE keyword on the START request. With the USERID keyword, for example, you could issue the following request on system MVS01:

```
ATBTRACE START DSN(TRACE.DS) LU(LUA) TP(TPA) USERID(GOODWIN)
```

In this case, APPC/MVS on the MVS01 system compares not only these LU name and TP name values, but also the user ID value, with those specified (explicitly or implicitly) on both inbound and outbound Allocate requests. So the trace data you collect consists only of those conversations for which the LU, TP, and user ID values for the START and Allocate requests match.

If a TP allocates a conversation with TPA, specifying a security type with security fields that LUA cannot support, the system downgrades the Allocate request by stripping out the security fields that are not supported. For example, if a TP issues the Allocate request with security_same, but the APPL definition statement for the LUA specifies CONV or NONE, the system strips out the user ID value specified on the Allocate request. In this case, APPC/MVS does not trace the conversations because the user ID value on the START request (user ID GOODWIN) does not match the user ID value on the inbound Allocate request (the user ID is missing).

If you want to make sure you are collecting trace data for inbound conversations for which the user ID is not passed, enter another START request for the LU/TP combination, this time specifying the SECNONE keyword instead of USERID. If you do so, APPC/MVS traces all conversations with matching LU/TP values and a Security_type of security_none, along with conversations with matching LU/TP/USERID values. Even though the resulting trace data might contain information for some extraneous security_none conversations, you probably have less trace data to sort than if you used the default of tracing all conversations for the LU/TP combination.

Collecting Trace Data for Security_None Conversations

The scenario in “Filtering Trace Data by User ID” on page 91 illustrates only one possible use for the SECNONE keyword; you also may use it when your only interest is in conversations with a Security_type of security_none. Just as the USERID keyword allows you to filter trace data by a particular user ID, specifying the SECNONE keyword for an LU/TP combination allows you to collect trace data for only those inbound and outbound conversations that are established with a Security_type of security_none.

Collecting Trace Data for TPs with Multiple Levels

In Figure 31 on page 92, notice that TPA is defined in more than one TP profile file, with different TP levels. Because APPC/MVS does not provide the ability to identify which level of TP you want to trace, APPC/MVS traces all conversations established for the LUA/TPA combination, regardless of the TP level. For example, TPB allocates to a user-level instance of TPA (2 and 2a), whereas TPC allocates to a system-level instance of TPA (3 and 3a).

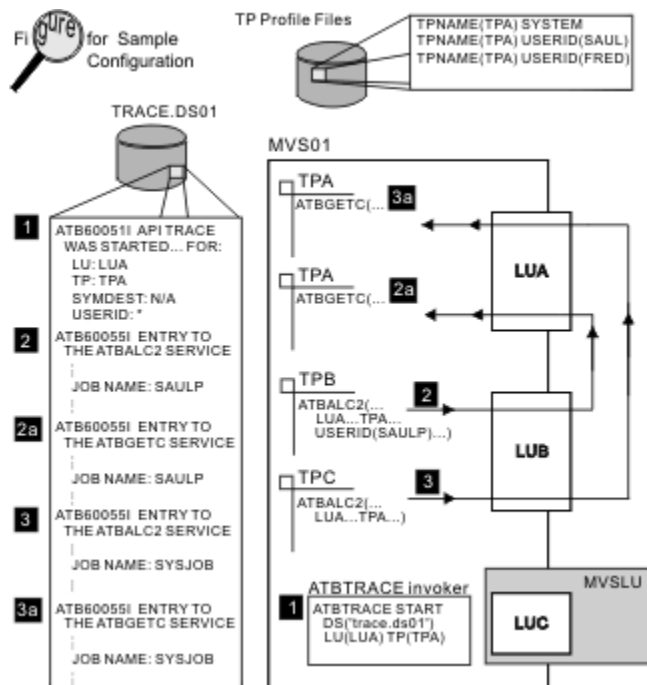


Figure 31. Collecting API Trace Data for TPs with Multiple Levels

The trace data set contains API trace data for both levels, which might make interpreting the trace data more difficult, especially if TPA's processing varies by level. See [“Determining the Level of TP Traced”](#) on page 107 for advice on distinguishing trace data for different levels of the TP.

Some Suggestions for ATBTRACE START Requests...

Given the scenarios in [“How APPC/MVS Handles an ATBTRACE START Request”](#) on page 86, issuing ATBTRACE START requests might seem like tedious work, but how much you have to do depends on the circumstances. For example, if you know that only one user is experiencing problems with TPA's processing, entering one START request with the USERID keyword is probably sufficient to collect all the pertinent trace data. However, if TPA's processing is dependent on correct processing between TPB and TPC, you might need to enter additional START requests for TPB or TPC as well, to collect trace data to determine whether the problem exists in the conversation between TPB and TPC.

Together with your knowledge of the TPs' processing and the symptoms that users experience, the flexibility that the API trace facility provides can help you more easily and efficiently start collecting trace data. The following list includes some suggestions that might help you decide how to use the API trace facility:

- If your installation is using VTAM generic resource support for APPC/MVS LUs, use the generic resource name as the value for the LU keyword on the ATBTRACE START request. This capability means that, with only one START request, you can collect trace data when an inbound Allocate request for the specified TP arrives at any of the LUs in the generic resource group. (If the group spans systems, however, you must enter the START request on each system with a member of the generic resource group.)

z/OS MVS Planning: APPC/MVS Management contains possible security requirements related to using the API trace facility, and general information about VTAM generic resource support for the installation.

- Consider invoking the ATBTRACE exec through TP profile JCL when you want to collect trace data each time the TP runs. Using TP profile JCL ensures that the START request is issued before the conversation is completely established; however, if another instance of the TP is allocated while the first instance is still running, the subsequent ATBTRACE START request will fail because the trace data set is already in use by another user.
- In a production environment, when speed and efficiency are critical:
 - Use TSO/E implicit or explicit invocations to start tracing. These invocation methods are ideally suited for dynamically controlling tracing activity for only a few LU/TP combinations. These methods allow you to minimize performance impact by limiting tracing activity to specific LU/TP combinations, and by limiting the amount of time tracing is active.
 - If your installation is experiencing problems with multiple TPs on multiple systems, you might start traces through a high-level language program or batch job, rather than issuing multiple START requests through TSO/E. However, you could still easily stop tracing activity through TSO/E, regardless of the method you used to start tracing.

Using the ATBTRACE REXX Exec

The interface to the API trace facility is the ATBTRACE REXX exec. Through the ATBTRACE exec, you control API tracing activity by issuing START and STOP requests, or obtain tracing status by issuing LIST requests.

The following topics describe:

- The programming requirements for and output of the ATBTRACE REXX exec, which is the programming interface for the API trace facility
- The methods of invoking the ATBTRACE REXX exec, and examples of each
- The syntax of and parameter descriptions for each type of ATBTRACE request.

Programming Considerations

Except where noted in the following topics, the environmental requirements, restrictions, and return codes are the same for all types of ATBTRACE request, regardless of the method you use to invoke the ATBTRACE REXX exec.

Requirements

Minimum authorization:	Problem state with any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller cannot hold any locks

Restrictions

- For all types of ATBTRACE requests, the caller cannot have any enabled unlocked (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.
- Depending on your installation's security policy, START and STOP requests might require READ access to the ATBTRACE security profile. If APPC/MVS returns an ATB6xxxxI message indicating security violations, see your system programmer or system administrator for assistance.

Output from the API Trace Facility

The API trace facility produces:

- Return codes to indicate successful or unsuccessful processing of an ATBTRACE START, STOP, or LIST request
- Status or error messages to provide additional information to issuers of START, STOP, or LIST requests
- Trace data in the form of ATB6xxxxI messages.

The following sections provide details about return codes and status or error messages; [Chapter 10, “API Trace Facility Messages,”](#) on page 291 describes all of the messages for the API trace facility.

Return Codes

Hexadecimal Return Code	Meaning
0	<p>The ATBTRACE START, STOP, or LIST request was processed successfully. Additional results depend on the type of request:</p> <ul style="list-style-type: none"> • For a START request, APPC/MVS will collect trace data for the LU/TP combination (by optional user ID, if the USERID keyword was specified on the request), for conversations established after ATBTRACE request processing completed. • For a STOP request, APPC/MVS stopped all active traces for the data set specified on the request, and closed the trace data set. • For a LIST request, APPC/MVS returned the list results for the data set specified on the request, or for all trace data sets in this system. The location of the list results depends on the method used to invoke the ATBTRACE REXX exec.
8	<p>The ATBTRACE START, STOP, or LIST request was not processed successfully, for the reason indicated by an accompanying ATB600xxI message. The location of the ATB message depends on the method used to invoke the ATBTRACE REXX exec.</p>

Messages

In addition to writing trace data in the form of ATB6xxxxI messages, the API trace facility also issues messages to report:

- Successful and unsuccessful completion of an ATBTRACE START, STOP, or LIST request.
- Syntax errors in a START, STOP, or LIST request.
- Incorrect programming environment for ATBTRACE invokers.
- Incorrect security authorization for START and STOP requests.
- 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.

The method you use to invoke an ATBTRACE request determines where APPC/MVS returns these status and error messages; see sections of [“Methods of Invoking the ATBTRACE REXX Exec”](#) on page 94 for details.

Methods of Invoking the ATBTRACE REXX Exec

To start, stop, or list API tracing activity, you may invoke the ATBTRACE REXX exec:

- In the TSO/E foreground, explicitly or implicitly, through the EXEC command
- In MVS batch, through JCL

- Through the JCL in an APPC/MVS TP profile
- From a high-level language program.

Which method you choose depends on the circumstances. You do not need to issue all requests through the same method; for example, you can start API tracing through JCL, but stop that tracing activity by issuing the TSO/E EXEC command. [“Some Suggestions for ATBTRACE START Requests...”](#) on page 92 might help you decide which methods to use; individual descriptions of the START, STOP, and LIST requests contain additional examples of circumstances for selecting a particular invocation method.

The following sections contain APPC/MVS-specific requirements and examples for each method of invoking the ATBTRACE REXX exec. Depending on the method, you might have to refer to additional sources:

- [z/OS TSO/E REXX User's Guide](#) for general information about invoking execs in the foreground, in batch, and from a high-level language program.
- [z/OS TSO/E REXX Reference](#) for general programming requirements (such as register contents on entry) for invoking execs in batch and from programs.
- [z/OS MVS Planning: APPC/MVS Management](#) for information about TP profile files.
- [z/OS MVS JCL Reference](#) for information about JCL.

Invoking the ATBTRACE Exec in the TSO/E Foreground

Use the TSO/E EXEC command processor to either explicitly or implicitly invoke the ATBTRACE REXX exec in the TSO/E foreground. The EXEC command runs non-compiled programs in TSO/E.

The ATBTRACE exec displays ATB6xxxxI status and error messages on the terminal from which you invoked the exec.

Unlike APPC/MVS callable services, ATBTRACE requests issued from TSO/E are not protected from attention interrupts, so you can interrupt the trace request processing before APPC/MVS completes it. APPC/MVS defers the interrupt only until processing progresses to a point at which the interrupt will not cause contention for a system resource.

Explicitly Invoking the ATBTRACE Exec

To explicitly invoke the ATBTRACE exec, enter the EXEC command followed by the name of the data set that contains the exec, SYS1.SBLSCLI0, followed by parameters enclosed in single quotes and the keyword "exec". For example:

- To start API tracing, specifying a fully qualified data set name for the trace data set, enter on the command line:

```
EXEC 'SYS1.SBLSCLI0(atbtrace)'
```

```
'start dataset(''userx.trace.data'') lu(lux) tp(tpx) userid(userx)' exec
```

Note that you must enclose a fully qualified trace data set name within two pairs of single quotes.

- To start API tracing, specifying an unqualified data set name, enter on the command line:

```
EXEC 'SYS1.SBLSCLI0(atbtrace)'
```

```
'start dataset(trace.data) lu(lux) tp(tpx) userid(userx)' exec
```

The trace data set name will be prefixed with a high-level qualifier that is the user ID of the invoker of the ATBTRACE exec.

Implicitly Invoking the ATBTRACE Exec

To implicitly invoke the ATBTRACE exec, perform the following steps:

1. Allocate the SYS1.SBLSCLI0 data set to the system file SYSEXEC or SYSPROC. For example:


```
alloc f(sysexec) da('SYS1.SBLSCLI0')
```

2. Enter the exec name, followed by its parameters, on the command line:

```
atbtrace start dataset('userx.trace.data') lu(lux) tp(tpx) userid(userx)
```

Note that, unlike the explicit invocation, you need only enclose a fully qualified trace data set name within single quotes. However, an unqualified data set name is treated the same way for explicit and implicit invocations: you specify an unqualified name without enclosing it in any quotes, and the name will be prefixed with a high-level qualifier that is the user ID of the invoker of the ATBTRACE exec.

Invoking the ATBTRACE Exec in MVS Batch

To run the ATBTRACE exec in batch mode, you have two choices: using IRXJCL to run ATBTRACE in the TSO/E background, or using IKJEFT01 to run ATBTRACE in a non-TSO/E address space. For either choice:

- You need to include a SYSEXEC DD statement in the JCL to specify SYS1.SBLSCLI0, the data set that contains the ATBTRACE exec
- ATB6xxxxI status and error messages, and information resulting from an ATBTRACE request, are written to the output device specified through the SYSTSPRT DD statement (either the invoker's job log, or a data set)
- API trace data entries are written to the data set specified through the DATASET parameter on ATBTRACE.

Using IRXJCL

To invoke the ATBTRACE exec using IRXJCL (to run the exec in a non-TSO/E address space), specify the following on the JCL EXEC statement:

1. IRXJCL on the PGM parameter, and
2. ATBTRACE and its parameters on the PARM parameter.

For example, the following contains JCL to submit an ATBTRACE LIST request for active traces for all trace data sets on the system:

```
//JOEA    JOB MSGLEVEL=(1,1)
//*
/* EXAMPLE OF LISTING ALL TRACES BY INVOKING
/* ATBTRACE LIST USING IRXJCL
/*
/* LIST OF ALL THE API TRACES WILL BE WRITTEN
/* EITHER TO JOBLOG
/*   SYSTSPRT DD SYSOUT=A
/* OR TO USER SPECIFIED DATA SET
/*   SYSTSPRT DD DSN=XX.YY
/*
/*
/*TRACE   EXEC PGM=IRXJCL,
/*        PARM='ATBTRACE LIST'
//SYSTSPRT DD   DSN=JOE.APILIST.DATASET,DISP=OLD
//SYSEXEC DD   DSN=SYS1.SBLSCLI0,DISP=SHR
/*
```

Another example contains JCL to submit an ATBTRACE LIST request for active traces of only one trace data set:

```
//JOEA    JOB MSGLEVEL=(1,1)
//*
/* EXAMPLE OF LISTING TRACES FOR A SINGLE DATA SET
/* ATBTRACE LIST USING IRXJCL.
/*
/* LIST OF ALL THE API TRACES STARTED WITH THE
/* DATA SET SUPPLIED ON ATBTRACE LIST REQUEST
/* WILL BE WRITTEN
/* EITHER TO JOBLOG
/*   SYSTSPRT DD SYSOUT=A
/* OR TO USER SPECIFIED DATA SET
/*   SYSTSPRT DD DSN=XX.YY
```



```

/*
/*
//TRACE   EXEC PGM=IRXJCL,
//        PARM='ATBTRACE LIST DS(''JOE.TRACE'')'
//SYSTSPRT DD   DSN=JOE.APILIST.DATASET,DISP=OLD
//SYSEXEC  DD   DSN=SYS1.SBLSCLI0,DISP=SHR
/*

```

Using IKJEFT01

To invoke the ATBTRACE exec using IKJEFT01 (to run the exec in the TSO/E background), specify the following on the JCL EXEC statement:

1. IKJEFT01 on the PGM parameter, and
2. ATBTRACE and its parameters either:
 - On the PARM parameter, or
 - Through explicit or implicit use of the EXEC command in the input stream.

For example, the following contains JCL to submit an ATBTRACE STOP request:

```

//JOEA    JOB MSGLEVEL=(1,1)
/*
/*
/* EXAMPLE OF STOPPING A TRACE BY INVOKING
/* ATBTRACE IN BATCH USING IKJEFT01 WHICH
/* WILL BRING UP A TSO/E ENVIRONMENT VIA
/* THE TMP
/*
/*
/* MESSAGES FROM THE REXX EXEC WILL BE WRITTEN
/* EITHER TO JOBLLOG
/* SYSTSPRT DD SYSOUT=A
/* OR TO USER SPECIFIED DATA SET
/* SYSTSPRT DD DSN=XX.YY
/*
/*
/*
//TRACE   EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC DD   DSN=SYS1.SBLSCLI0,DISP=SHR
//SYSTSPRT DD   SYSOUT=A
//SYSTSIN  DD   *
//          %ATBTRACE STOP DATASET('JOE.TRACE')
/*

```

Invoking the ATBTRACE Exec from an HLL Program

To invoke the ATBTRACE exec from a high-level language (HLL) program, code the program to load and call the IRXJCL program, passing a parameter list that contains ATBTRACE and its parameters.

ATB6xxxxI status and error messages, and information resulting from an ATBTRACE request, are written to the output device specified through the SYSTSPRT DD statement (either the invoker's job log, or a data set). API trace data entries are written to the data set specified through the DATASET parameter on ATBTRACE.

For example, the following PL/I program, JCLXMP1, uses IRXJCL to invoke the ATBTRACE exec to start tracing the LU/TP combination LUX and TPX:

```

JCLXMP1 : Procedure Options (Main);
/* Function: Call a REXX exec from a PL/I program using IRXJCL          */
/* Note: This example is for PL/I Version 2.                             */

DCL IRXJCL EXTERNAL OPTIONS(RETCODE, ASSEMBLER);
DCL 1 PARM_STRUCT, /* Parm to pass to IRXJCL */
      5 PARM_LNG BIN FIXED (15), /* Length of the parameter */
      5 PARM_STR CHAR (53); /* String passed to IRXJCL */
DCL PLIRETV BUILTIN; /* Defines the return code built-in */
PARM_LNG = LENGTH(PARM_STR); /* Set the length of string */
/*
/* PARM_STR = 'ATBTRACE START DATASET(A.B) LU(LUX) TP(TPX) USERID(*)';/*
/* Set string value to the exec to
/* be invoked followed by the exec's
/* arguments
/*
/* FETCH IRXJCL; /* Load the address of entry point */
/* CALL IRXJCL (PARM_STRUCT); /* Call IRXJCL to execute the REXX
/* exec and pass the argument */

```

```

PUT SKIP EDIT ('Return code from IRXJCL was:',PLIRETV) (a, f(4));
/* Print out the return code from
   the exec ATBTRACE
END ;
/* End of program

```

The following JCL runs program JCLXMP1:

```

//USERID JOB MSGLEVEL=(1,1)
//*
//* EXAMPLE OF INVOKING ATBTRACE USING A HIGH
//* LEVEL PROGRAM SUCH AS PL/I.
//*
//* MESSAGES FROM THE REXX EXEC WILL BE WRITTEN
//* EITHER TO JOBLOG
//* SYSTSPRT DD SYSOUT=A
//* OR TO USER SPECIFIED DATA SET
//* SYSTSPRT DD DSN=XX.YY
//*
//*
//TRACE EXEC PGM=JCLXMP1
//SYSTSPRT DD DSN=USERID.MESSAGE.DATASET,DISP=OLD
//SYSEXEC DD DSN=SYS1.SBLSCLI0,DISP=SHR
/*

```

Invoking the ATBTRACE Exec from TP Profile JCL

To invoke the ATBTRACE exec from the JCL in a TP profile, specify the following on the JCL EXEC statement:

1. IRXJCL on the PGM parameter, and
2. ATBTRACE and its parameters on the PARM parameter.

Also include a SYSEXEC DD statement in the JCL to specify SYS1.SBLSCLI0, the data set that contains the ATBTRACE exec.

ATB6xxxxI status and error messages, and information resulting from an ATBTRACE request, are written to the output device specified through the SYSTSPRT DD statement (either the invoker's job log, or a data set). API trace data entries are written to the data set specified through the DATASET parameter on ATBTRACE.

The following example invokes the APPC/MVS administration utility to add a TP profile that contains JCL to:

- Invoke the ATBTRACE exec to start a trace for this TP,
- Run the inbound TP, and
- Invoke the ATBTRACE exec to stop tracing.

```

//TPADD JOB MSGLEVEL=(1,1)
//TPADD EXEC PGM=ATBSDFMU
//*****
//* EXAMPLE OF STARTING TRACE FROM TP PROFILE
//*
//* DEFINES A TP. ALL KEYWORDS ARE NOT SHOWN IN THIS EXAMPLE
//*
//* MESSAGES FROM THE REXX EXEC WILL BE WRITTEN
//* EITHER TO JOBLOG
//* SYSTSPRT DD SYSOUT=A
//* OR TO USER SPECIFIED DATA SET
//* SYSTSPRT DD DSN=XX.YY
//*
//* STARTS API TRACING (TRACE DATA SET IS QUALIFIED WITH QUOTES)
//* EXECUTES MYTPX PROGRAM
//* STOPS API TRACING (TRACE DATA SET IS QUALIFIED WITH QUOTES)
//*****
//SYSPRINT DD SYSOUT=*
//SYSSDLIB DD DSN=APPC.TP.DATASET,DISP=SHR
//SYSSDOUT DD SYSOUT=*
//SYSIN DD DATA
TPADD
TPNAME(TPX)
:
:
TPSCHED_DELIMITER(DLM1)

```

```

:
:
JCL_DELIMITER(DLM2)
//TPSTEP JOB 'TPNAME',MSGLEVEL=(1,1)
//STARTTR EXEC PGM=IRXJCL,
//          PARM='ATBTRACE START DATASET(''JOE.TRACE'') LU(LUX) TP(TPX)'
//SYSTSPRT DD SYSOUT=A
//SYSEXEC DD DSN=SYS1.SBLSCLI0,DISP=SHR
//*
//TPEXE EXEC PGM=MYTPX
//STEPLIB DD DSN=JOE.LOAD,DISP=SHR
//*
//STOPTR EXEC PGM=IRXJCL,
//          PARM='ATBTRACE STOP DATASET(''JOE.TRACE'')'
//SYSTSPRT DD SYSOUT=A
//SYSEXEC DD DSN=SYS1.SBLSCLI0,DISP=SHR
//*
DLM2
DLM1
/*

```

Starting an API Trace

Through the ATBTRACE REXX exec, you may start an API trace for a particular LU/TP combination, or LU/TP/USERID combination, specifying a data set to contain the trace entries. Depending on the way your installation chose to set up API trace data sets, to successfully issue a START request, you must:

- Specify a pre-allocated, sequential data set for the trace data
- Have the appropriate access to the data set.

If the data set is in use already, you must be able to either specify another data set, or use the same user ID from which the first START request was issued for this data set. If errors occur during START processing because of problems with the data set, with your access authority, or with the API trace facility itself, APPC/MVS returns an ATB6xxxxI message to inform the invoker of the START request.

When it successfully processes a START request, APPC/MVS returns message ATB60035I to the issuer of the request to indicate successful completion, and writes ATB60051I in the specified data set. This message contains a user ID and time stamp to indicate who started this trace for this data set, and at what time.

Although coding the STOP and LIST requests is fairly simple, determining the values to specify when coding START requests can be complicated because of a number of factors. See [“Starting API Tracing Activity” on page 86](#) for examples and suggestions for starting API traces.

Selecting the Invocation Method for the ATBTRACE START Request

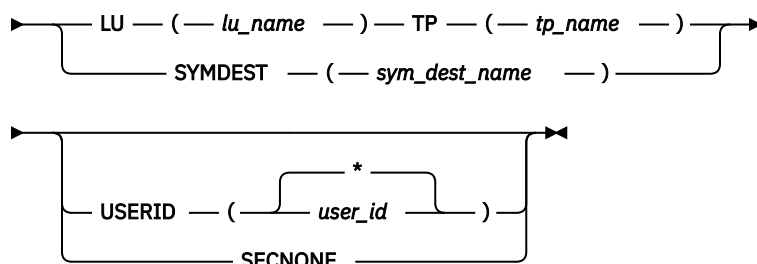
To start API tracing activity, you may use any of the invocation methods described in [“Methods of Invoking the ATBTRACE REXX Exec” on page 94](#); reviewing the examples and suggestions in [“Starting API Tracing Activity” on page 86](#) might help you select an appropriate method. Before issuing a START request, make sure you are aware of the programming environment and restrictions described in [“Programming Considerations” on page 93](#).

Coding an ATBTRACE START Request

Use the following syntax diagram and parameter descriptions to code an ATBTRACE request to start tracing a conversation. Remember that, depending on the TPs involved in the conversation you want to trace, you might have to issue more than one ATBTRACE request or also invoke a trace facility on an operating system other than MVS.

Syntax

►► ATBTRACE — START — DATASET — (— *ds name* —) ►



Parameters

START

Specifies that API tracing is to start for the conversation identified by the other parameters on the START request. APPC/MVS traces only those conversations established after it successfully processes the START request, when the inbound or outbound call to establish the conversation explicitly or implicitly uses values for LU and TP (and possibly user ID) that exactly match those specified on the START request.

DATASET(*ds name*)

Specifies the data set that is to contain the trace data for the conversations to be traced. You may use the abbreviations DA, DSNAME, DSN, or DS instead of DATASET.

ds_name can be either the fully qualified or unqualified name of a pre-allocated, sequential data set. A fully qualified name must be specified within single quotes. When the data set name is specified without quotes, APPC/MVS adds a high-level qualifier, the user ID of the ATBTRACE invoker, to the data set name.

When ATBTRACE is invoked in MVS batch mode, you must specify a fully qualified data set name; otherwise, APPC/MVS rejects the ATBTRACE request.

LU(*lu_name*) TP(*tp_name*)

Specifies the LU and TP combination that APPC/MVS is to trace. When both of the following conditions are true, APPC/MVS collects trace data for the conversation:

- The LU keyword value matches the partner LU value passed (explicitly or through a symbolic destination name) on the APPC/MVS Allocate, Register_for_Allocates, or CPI-C Set_Partner_LU_Name service.
- The TP keyword value matches the partner TP value passed (explicitly or through a symbolic destination name) on the APPC/MVS Allocate, Register_for_Allocates, or CPI-C Set_TP_Name service.

lu_name is a 17-character field containing one of the following:

- A network-qualified LU name, in the form `network_id.network_LU_name`, in which `network_id` is the 8-byte ID of the network, and `network_LU_name` is the 8-byte local LU name
- An 8-byte local LU name
- A VTAM generic resource name.

tp_name is a 64-byte character field containing the name of a standard TP, a multi-trans TP, a TP registered for test or a served TP. *TP_name* must exactly match the name specified on Allocate requests, including case. *TP_name* cannot be an SNA service TP name; APPC/MVS does not support tracing of SNA service TPs.

The LU and TP keyword combination is mutually exclusive with the SYMDEST keyword. You must specify either the SYMDEST keyword or the LU and TP keyword combination to start tracing.

SYMDEST(sym_dest_name)

Specifies, through a symbolic destination name, the LU and TP combination that APPC/MVS is to trace. When either of the following conditions are true, APPC/MVS collects trace data for the conversation:

- The SYMDEST keyword value matches the value for the Sym_dest_name parameter passed on the APPC/MVS Allocate, Register_for_Allocates, or CPI-C Initialize_Conversation service.
- The specific LU and TP names in the side information entry (identified through *sym_dest_name*) match the values for the Partner_LU and TP_name parameters passed on the APPC/MVS Allocate or Register_for_Allocates service, or the CPI-C Set_Partner_LU and Set_TP_Name services.

sym_dest_name is an 8-byte character field containing the symbolic name of the destination LU and partner TP.

This keyword is mutually exclusive with the LU and TP keyword combination. You must specify either the SYMDEST keyword or the LU and TP keyword combination to start tracing.

USERID(*|user_id)

Specifies an additional filter to limit the conversations traced for a specific LU and TP combination. You may use the abbreviations USER or U instead of USERID. Use this keyword if you want to collect trace data for inbound and outbound conversations established only by a particular user, for this LU/TP combination.

user_id is a 10-byte character field containing the user ID to be used as a filter. APPC/MVS traces the conversation only when the value for USERID matches the value of the User_id parameter specified on the Allocate service. If the conversations you want to trace might be allocated without any user ID, start the trace using the SECNONE keyword instead of USERID.

The USERID and SECNONE keywords are both optional, and are mutually exclusive. If you do not specify either USERID or SECNONE, APPC/MVS uses the default value for USERID, '*', which means that all conversations established for the LU/TP combination are traced, even conversations with a Security_type of security_none.

SECNONE

Specifies that APPC/MVS is to trace only conversations allocated without a user ID specified, for this LU and TP combination. When either of the following conditions is true, APPC/MVS collects trace data for the conversation:

- A Security_type of security_none is specified on the Allocate service
- A Security_type of security_same or security_pgm is specified on the Allocate service, but VTAM downgraded the conversation to security_none.

The USERID and SECNONE keywords are both optional, and are mutually exclusive. If you do not specify either USERID or SECNONE, APPC/MVS uses the default value for USERID, '*', which means that all conversations established for the LU/TP combination are traced, even conversations with a Security_type of security_none.

Stopping Trace Activity

Through the ATBTRACE REXX exec, you may stop all active API traces for a particular data set. Depending on the way your installation chose to set up API trace data sets and restrict their use, you might be stopping traces started by other programmers when you issue a STOP request. Before issuing the STOP request, consider requesting a list of active traces for this data set, and reviewing the results to determine whether you might be adversely affecting the work of others. Keep in mind that, after an error occurs, the longer you wait to stop tracing, the greater the risk of losing pertinent trace data through wrapping.

Your installation might restrict the use of not only API trace data sets, but also APPC/MVS LUs and TPs. To successfully issue a STOP request, you must either:

- Have issued the first ATBTRACE START request for this data set, or
- Have the appropriate access to the security profile for the ATBTRACE resource for all the LUs and TPs for which APPC/MVS is storing trace data in this data set.

Otherwise, APPC/MVS rejects the STOP request, issuing an ATB6xxxxI message that states the security violation.

When it successfully processes a STOP request, APPC/MVS returns message ATB60036I to the issuer of the request to indicate successful completion, and writes ATB60052I in the specified data set. This message contains a user ID and time stamp to indicate who stopped the traces for this data set, and at what time.

In some cases, APPC/MVS must delay the processing of a STOP request, usually because of extensive I/O activity. If a delay is necessary, APPC/MVS issues message ATB60024I to indicate that the STOP request is queued for processing. To determine when the STOP request has completed, issue an ATBTRACE LIST request for this data set; if the response to the LIST request is ATB60047, the STOP request is still pending.

If errors occur during STOP processing, APPC/MVS issues various ATB6xxxxI messages to inform the issuer of the STOP request.

Selecting the Invocation Method for the ATBTRACE STOP Request

To stop API tracing activity, you may use any of the invocation methods described in “Methods of Invoking the ATBTRACE REXX Exec” on page 94, regardless of the method you chose to start an API trace. Before issuing a STOP request, make sure you are aware of the programming environment and restrictions described in “Programming Considerations” on page 93.

In general, using the same method to both start and stop tracing is the easiest approach, but unusual circumstances might require mixing methods. For example, suppose a particular TP is not functioning correctly in your installation's production system. You might choose to issue the START and STOP requests in the TP profile JCL, so you can collect trace data each time the TP runs. If you find that, because multiple instances of the TP are running, tracing activity is impacting the performance of APPC/MVS work, you could issue the STOP request through TSO/E instead.

Coding an ATBTRACE STOP Request

Use the following syntax diagram and parameter descriptions to code an ATBTRACE request to stop tracing a conversation.

➡ ATBTRACE — STOP — DATASET — (— *ds_name* —) ➡

Parameters

STOP

Specifies that all API tracing activity for the specified data set is to stop.

DATASET(*ds_name*)

Specifies the data set for which tracing activity is to stop. You may use the abbreviations DA, DSNAME, DSN, or DS instead of DATASET.

ds_name can be either the fully qualified or unqualified name of a pre-allocated, sequential data set. A fully qualified name must be specified within single quotes. When the data set name is specified without quotes, APPC/MVS adds a high-level qualifier, the user ID of the ATBTRACE invoker, to the data set name.

When ATBTRACE is invoked in MVS batch mode, you must specify a fully qualified data set name; otherwise, APPC/MVS rejects the ATBTRACE request.

Listing Active API Traces

Through the ATBTRACE REXX exec, you may request a list of all active API traces for a particular data set, or for all API trace data sets on the system. When it successfully processes a LIST request, APPC/MVS returns message ATB60046I to the issuer of the request; this message contains:

- A user ID and time stamp to indicate who issued the first START request for this data set, and at what date and time.

- For each active trace, the parameter values that were specified on the START request, and the date and time at which the START request was issued.

Because an unlimited number of traces may be active for each trace data set, you might have difficulty finding the end of the list. Look for message ATB60042I, which APPC/MVS returns when LIST processing is complete.

APPC/MVS does not return LIST information for a data set for which STOP processing is in progress. Depending on the type of LIST request you specify (one data set or all), APPC/MVS returns message ATB60047I when STOP processing is in progress:

- If you specified one data set, you get only ATB60047I in response
- If you specified all, you get the same message for each data set for which STOP processing is in progress, along with the list of active traces for the remaining trace data sets on the system.

Selecting the Invocation Method for the ATBTRACE LIST Request

To list API tracing activity, you may use any of the invocation methods that are valid for START and STOP requests. The method you choose determines where APPC/MVS returns the list results. Invocation methods and output destinations are described in [“Methods of Invoking the ATBTRACE REXX Exec” on page 94](#). Before issuing a LIST request, make sure you are aware of the programming environment and restrictions described in [“Programming Considerations” on page 93](#).

Coding an ATBTRACE LIST Request

Use the following syntax diagram and parameter descriptions to code an ATBTRACE request to list all active API traces for a specified data set, or for all API trace data sets in use on this system.



Parameter Description

LIST

Requests a list of all active API traces for a specified data set, or for all API trace data sets in use on this system.

DATASET(ds_name)

Specifies the API trace data set for which a list of active traces is requested. You may use the abbreviations DA, DSN, or DS instead of DATASET.

ds_name can be either the fully qualified or unqualified name of a pre-allocated, sequential data set. A fully qualified name must be specified within single quotes. When the data set name is specified without quotes, APPC/MVS adds a high-level qualifier, the user ID of the ATBTRACE invoker, to the data set name.

When ATBTRACE is invoked in MVS batch mode, you must specify a fully qualified data set name; otherwise, APPC/MVS rejects the ATBTRACE request.

Interpreting API Trace Data

The API trace facility writes trace data in the form of ATB6xxxxI messages; illustrates the format and explains the contents of each message. These messages are stored in a particular API trace data set, in a format that you can view with an editor or browser, once all tracing activity for the data set has stopped.

For a particular LU and TP (and possibly USERID) combination, these ATB6xxxxI messages (or trace entries) document:

- ATBTRACE START and STOP requests, including all parameters and their values specified for each request.
- The contents of FMH-5 records exchanged between conversing TPs, excluding passwords.

- Both entry to and return from a supported APPC/MVS or CPI-C service.
- The number of trace entries lost if APPC/MVS had to temporarily suspend tracing activity.

Depending on the location of the TP's partners, you might have to compare the contents of more than one trace data set; perhaps comparing the API trace data with trace data from a facility provided on a platform other than MVS. The format of API trace data is similar to that provided by CM/2 on OS/2, so you can more easily diagnose problems between APPC/MVS TPs and their partners on OS/2.

Reading Service-Entry and Service-Return Trace Entries

For synchronous calls, the API trace facility provides one trace entry for entry to a particular service, and one for return from that service. For asynchronous calls, API trace provides three trace entries: one for entry to the service, one when asynchronous processing of the service begins, and the last when asynchronous service processing completes.

For both entry and return trace entries, some of the supplied and returned parameter values have been converted into text, so anyone using the trace data for debugging can more easily determine the intent and result of the service call. For example, on an Allocate call, suppose a TP specifies a value of 0 for the Conversation_type parameter, which represents a basic, rather than a mapped, conversation. In the service-entry trace entry, the value for Conversation_type appears as the phrase "BASIC_CONVERSATION" rather than 0.

For both synchronous and asynchronous calls, the service-entry trace entry includes a list of only those parameters for which the caller had to supply values. The service-return trace entry includes a list of only those parameters for which APPC/MVS returned values. The service-return entry might also contain error information, if an error occurred during service processing.

To reduce the amount of data in the trace data set, the API trace facility includes only the first few bytes and last few bytes of send or receive data.

The following example contains sample API trace data for a successful, synchronous call to the Allocate service:

```
ATB60055I ENTRY TO THE ATBALC2 SERVICE:
TIMESTAMP: 07/20/1995 03:05:36.632718
ASID: 0045
TCB ADDR: 00550324
JOB NAME: JOEA
LU: NET1.LUA
TP: TPA
USERID: JOE
CONVID: 0000000000000000
PARAMETERS:
  CONVERSATION_TYPE: BASIC_CONVERSATION
  SYM_DEST_NAME:
  PARTNER_LU_NAME: LUA
  MODE_NAME: TRANPAR
  TP_NAME_LENGTH: 3
  TP_NAME: TPA
  RETURN_CONTROL: WHEN_SESSION_ALLOCATED
  SYNC_LEVEL: CONFIRM
  SECURITY_TYPE: SEC_PGM
  USER_ID: MYUSER
  PASSWORD: *****
  PROFILE: MYGROUP
  USER_TOKEN: 0040
  NOTIFY_TYPE: 0000000000000000
  TP_ID: 0000000000000000
  LOCAL_LU_NAME:

ATB60056I THE ATBALC2 SERVICE COMPLETED.
TIMESTAMP: 07/20/1995 03:05:45.816390
ASID: 0045
TCB ADDR: 00550324
JOB NAME: JOEA
LU: NET1.LUA
TP: TPA
USERID: JOE
CONVID: 0345056400000005
PARAMETERS:
```



```
CONVERSATION_ID: 0345056400000005
RETURN_CODE: OK
```

Note that this sample trace entry includes MVS-specific information such as address space ID, TCB address, job name, and so on. This information can help you sort trace data if you are concurrently tracing multiple TPs or users, and storing all trace data in the same data set.

The following example contains sample API trace data for a successful, asynchronous call to the Allocate service:

```
ATB60055I ENTRY TO THE ATBALC2 SERVICE:
TIMESTAMP: 07/20/1995 03:06:36.632718
ASID: 0045
TCB ADDR: 00553324
JOB NAME: FRED
LU: NET1.LUA
TP: TPA
USERID: FRED
CONVID: 0000000000000000
PARAMETERS:
  CONVERSATION_TYPE: BASIC_CONVERSATION
  SYM_DEST_NAME: MYSYM
  PARTNER_LU_NAME:
  MODE_NAME:
  TP_NAME_LENGTH: 0
  TP_NAME:
  RETURN_CONTROL: WHEN_SESSION_ALLOCATED
  SYNC_LEVEL: CONFIRM
  SECURITY_TYPE: SEC_PGM
  USER_ID: MYUSER
  PASSWORD: *****
  PROFILE: MYGROUP
  USER_TOKEN: 0040
  NOTIFY_TYPE: 0000000107554344
  TP_ID: 0000000000000000
  LOCAL_LU_NAME:

ATB60057I SYNCHRONOUS RETURN FROM THE ATBALC2 SERVICE.
TIMESTAMP: 07/20/1995 03:06:45.872718
ASID: 0045
TCB ADDR: 00000000
JOB NAME: FRED
LU: NET1.LUA
TP: TPA
USERID: FRED
CONVID: 0345056400000005
PARAMETERS:
  RETURN_CODE: OK

ATB60056I THE ATBALC2 SERVICE COMPLETED.
TIMESTAMP: 07/20/1995 03:06:36.542735
ASID: 0045
TCB ADDR: 00000000
JOB NAME: FRED
LU: NET1.LUA
TP: TPA
USERID: FRED
CONVID: 0345056400000005
PARAMETERS:
  CONVERSATION_ID: 0345056400000005
  RETURN_CODE: OK
```

The following example contains sample API trace data for an unsuccessful call to the Send service, which fails because of a security violation detected by the partner LU:

```
ATB60055I ENTRY TO THE ATBSEND SERVICE:
TIMESTAMP: 07/20/1995 08:13:15.349258
ASID : 0045
TCB ADDR : 00553324
JOB NAME : FRED
LU : NET1.LUA
TP : TPA
USERID : MYUSER
CONVID : 049303F800000001
PARAMETER:
  CONVERSATION_ID : 049303F800000001
  SEND_TYPE : SEND_AND_CONFIRM
  SEND_LENGTH : 28
```

```
ACCESS_TOKEN      : 00000000
BUFFER            : 001CE3C8D9C5C540E3D6...40D4D94B40E2C3D6E3E3
NOTIFY_TYPE       : 00000000

ATB60056I TRACE DATA ON RETURN FROM THE ATBSEND SERVICE:
TIMESTAMP: 07/20/1995 08:14:15.349258
ASID             : 0045
TCB ADDR        : 00553324
JOB NAME        : FRED
LU              : NET1.LUA
TP              : TPA
USERID          : FRED
CONVID          : 049303F800000001
PARAMETER:
  RETURN_CODE: SECURITY_NOT_VALID
  ERROR_INFO:
    MESSAGE_TEXT_LENGTH      : 77
    MESSAGE_TEXT             : ATB80100I From VTAM macro APPCCMD:
    Primary error return code: 0004    secondary error return code: 0005
    sense code: 080F6051
    ERROR_LOG_PRODUCT_SET_ID_LENGTH: 14
    ERROR_LOG_PRODUCT_SET_ID     : .....MVSESA
    ERROR_LOG_INFORMATION_LENGTH : 120
    ERROR_LOG_INFORMATION        : ATB70017I TP security violation.
    Partner LU LUA rejected the allocate request because authorization
    checks failed.
```

Note that this sample trace entry contains additional diagnostic information about the security violation. This additional information is exactly what the Error_Extract service returns when a TP calls that service immediately after the previous call ended with an error.

If you know that your TP issues specific calls to services but trace entries for those service calls are missing (but not because of wrapping or suspension), the TP might be passing values that are not valid. APPC/MVS does not record a trace entry for a service call when either:

- The TP supplied a conversation ID on the call, but the conversation ID is not valid
- The TP supplied a conversation ID on the call, but the conversation was deallocated before the TP issued the call.

Also remember that, on the call to establish a conversation, the TP has to supply values for partner LU name, TP name, and possibly user ID that match those specified on the ATBTRACE START request. If the TP does not supply matching values, APPC/MVS does not generate any trace entries for the conversation.

Reading Trace Entries When Wrapping Occurred

Because you can issue more than one START request for the same data set, and because APPC/MVS might wrap data, you might find multiple START trace entries, but not necessarily a complete record of traces started for this data set. You might need to determine:

- Which entries are the most recent,
- Which entries might have been overwritten, or
- Which traces might have been active previously, but ended before the STOP request was issued.

To determine the completeness of the trace data, look for the STOP entry by searching for the character string “STOP”. This entry indicates the end of the most recent trace entries; any entries following the STOP entry are entries written before wrapping occurred. Also, the STOP entry itself lists all traces for this data set that were active when the STOP request was issued.

Finding All Trace Entries for a Specific Conversation

The API trace data set might contain trace entries for more than one conversation; because entries are arranged by time stamp, entries for the conversations are probably interleaved. Because most API trace entries contain the conversation ID, which is unique for each conversation, you can sort the trace entries by conversation ID to consolidate all the entries for a particular conversation. The only trace entries that do not contain such IDs are those for:

- APPC/MVS allocate queue services, and

- Services that do not directly affect the conversation, such as the APPC/MVS Get_TP_Properties service.

Determining the Level of TP Traced

Your installation can customize a TP's processing for different audiences by defining different TP profiles for the same TP, and assigning a different level in each TP profile key. On an ATBTRACE START request, you cannot specify the level of TP to be traced, so the API trace data set might contain a mix of trace entries from various levels of the TP. You might be able to restrict tracing activity to specific users through the USERID keyword on ATBTRACE; but, depending on how your installation defines different audiences, that might not be enough to ensure that you are collecting trace data for only one level of the TP. If you need to distinguish the TP's processing by determining the TP level, use the following information together to match trace data with the correct TP level:

- The LU, TP, user ID, and FMH-5 information from the API trace entries, and
- The GROUPID and USERID values in the appropriate TP profiles.

Assessing the Impact of Trace Entries Lost during Suspension

Depending on the volume of trace entries generated through TP processing, and possible contention for system resources, APPC/MVS might have to temporarily stop collecting trace data, so it can write the backlog of trace entries to the trace data set. When it resumes collecting trace data, APPC/MVS writes a trace entry that indicates how many potential trace entries were lost during the suspension.

This loss might or might not affect your ability to diagnose a problem, or to verify TP design or conversation flow. If your TP's design is relatively simple, and you can easily extract the existing trace entries for this TP from other entries in the data set, you might be able to use the existing entries to accomplish your task. Based on your knowledge of the TP's processing, and the type of information the API trace facility collects, you might be able to determine exactly what entries are missing, and determine their relative importance to the task at hand.

Otherwise, you might have to restart the trace for this TP to collect complete information about its processing. If you decide to restart the trace, consider the following actions that might help avoid another suspension:

- Restrict the trace as much as possible by altering the values you supply for the LU or USERID keywords on the ATBTRACE START request. For example, if you originally specified a generic resource name for the LU, you might have collected trace data for multiple instances of the TP, running on different LUs in the generic resource group. If you know the specific name of an LU in that group, you could specify that name instead of the generic resource name. Restricting the trace this way might reduce the volume of trace entries that APPC/MVS has to collect, and reduce the possibility of another suspension.
- Use a different data set for the trace entries by specifying a different data set name on the START request. Doing so might reduce or eliminate resource contention, and reduce the possibility of another suspension.

The lost entries might include a START entry for an ATBTRACE START request that was issued while tracing was suspended. By checking the LU/TP (and possibly user ID) combinations in trace entries with those on existing START entries, you can determine whether additional traces were started for this data set. STOP entries do appear in the trace data set, even if the STOP request was issued while tracing was suspended. In that case, APPC/MVS queues STOP requests to process once it has resumed tracing activity.

Assessing the Impact of Trace Entries Lost because of Termination

In addition to stopping API tracing activity in response to an ATBTRACE STOP request, APPC/MVS might have to stop active traces when it:

- Encounters a severe internal or I/O error while processing a START or STOP request
- Encounters an internal or I/O error while recording a trace entry
- Terminates normally or abnormally.

In these cases, APPC/MVS stops all active traces for the data set (or, for more global errors, all traces for all data sets), and rejects any subsequent ATBTRACE requests for the affected data set (or system). APPC/MVS also does not write any backlog of entries, so all outstanding data is lost. In fact, for I/O errors, the trace entries in the data set might be unusable.

Whenever possible, APPC/MVS writes a trace entry to indicate that it has stopped trace activity, and also notifies the operator by issuing a message to the console. Depending on the error that occurred, you might not find such an entry in the data set you're using, and the usual STOP entry won't appear either. To determine the end of the most recent trace entries, scan through the time stamps on the entry. If wrapping occurred, older trace entries appear after recent entries; otherwise, the most recent entries are at the end of the data set.

Overview of Error_Extract Service

Error_Extract is a TP conversation service that returns detailed information about errors indicated by return codes. Your TP can display the detailed error information to end users, or use it as input to a debugging program.

All error return codes from APPC/MVS indicate that one of the following error situations occurred in APPC processing:

- A TP called an APPC/MVS service with unknown values or incorrect parameters
- A TP called a service while a conversation is in a state that does not support the service
- APPC/MVS found an error that prevented the scheduling of a TP, such as:
 - An LU was either not defined or incorrectly defined
 - The address space in which a requestor TP is running did not have access to a scheduler
 - An internal error occurred.

IBM recommends that your TP call Error_Extract immediately after APPC/MVS returns a return code that indicates one of the errors listed above. Also, call Error_Extract only when errors occur in calls to supported services, which are listed in [Table 10 on page 81](#).

The following sections explain how to call Error_Extract and interpret the information that Error_Extract returns.

Types of Error Information that Error_Extract Returns

[Table 11 on page 108](#) shows the types of information that the Error_Extract service can return, the situations in which each type of information is returned, and a reference to the section in this document that describes that information:

<i>Table 11. Types of Information that Error_Extract Returns</i>		
Type	Error Situation	Reference
Error reason codes	APPC/MVS finds an error in the local TP or system	“Summary of Error_Extract Reason Codes” on page 313
Error messages	APPC/MVS finds an error in the local TP or system	“Error_Extract (ATB8) Messages” on page 343
Product set identifiers (IDs)	A remote system or TP finds an error in a conversation that involves a TP running on MVS	“Product Set Identifiers” on page 109
Error log information	A remote system or TP finds an error in a conversation that involves a TP running on MVS	“Error_Extract Error Log Information (ASB, ATB7) Messages” on page 317

An error reason code that is returned on a call to `Error_Extract` has the same meaning as the error message that is returned on the same call. [“Summary of Error_Extract Reason Codes” on page 313](#) maps the reason codes that `Error_Extract` returns with their associated messages. The following sections provide detailed explanations of the remaining two types of information that `Error_Extract` can return.

Error Log Information

A partner system might provide **error log information** to APPC/MVS to describe an error that occurs on that system. APPC/MVS can also send error log information to a partner system if it detects an error in a conversation that involves a TP running on MVS.

When APPC receives error log information, your TP can call `Error_Extract` to return that information in a convenient, readable format. When a partner TP or system is not APPC/MVS, error log information is available to `Error_Extract` only when the partner TP or system:

- Calls the CPI Communications `Set_Log_Data` call to specify the type of information that is to be logged
- Calls the `Send_Error` service to inform your TP that the partner system detected an error during a conversation
- Calls the `Deallocate` service with a `deallocate_type` of `CM_DEALLOCATE_ABEND` (to end the conversation abnormally)

`Error_Extract` can return only the first 512 bytes of error log information that is available.

The APPC/MVS scheduler or an alternate scheduler can also send error log information to a partner system or program. The information can describe errors that APPC/MVS finds when it tries to schedule a TP, which might help diagnose errors in the TP running on the partner system.

Product Set Identifiers

When a partner system or TP sends error log information to APPC/MVS, a *product set ID*, which identifies the hardware or software product set that is currently configured on the partner system, might be included. You can use the product set ID to identify the hardware or software product that found the specified error.

`Error_Extract` returns up to 256 bytes of the product set ID and the length of the ID to the caller. A length of zero indicates that the partner system or TP did not send a product set ID.

For information about the format of a product set ID, see the descriptions of the Product Set ID (X'10') and the Product Identifier (X'11') MS Common Subvectors in *SNA Formats*

Rules for Calling Error_Extract

Your TP can call `Error_Extract` *only*:

- For LU 6.2 TP conversation services and CPI Communications TP conversation calls (for a list of supported services, see [“Overview of Error_Extract Service” on page 108](#))
- For the *most recently completed call* to a conversation service (`Error_Extract` cannot return information for previous calls)
- For a conversation that was accepted or allocated by a TP that shares the same home address space as the caller.

When an error return code is returned to an APPC callable service, APPC/MVS retains the information that `Error_Extract` returns until one of the following occurs:

- Your TP calls the `Error_Extract` service
- The conversation is deallocated normally (the TP receives `deallocated_normal` return code)
- Your TP calls another APPC/MVS TP conversation service for the same conversation
- The APPC address space is cancelled or restarted.

You should design your TP so APPC/MVS retains the information that `Error_Extract` returns for as long as it is required.

Recommendation: Do not call another APPC/MVS service after receiving an error return code without first calling `Error_Extract`.

Calling `Error_Extract` for an Unestablished Conversation

If APPC/MVS cannot establish a conversation, APPC/MVS still assigns a conversation ID to the request to allocate the conversation. Your TP can use that conversation ID as input to `Error_Extract` when errors occur in calls to the following services:

- LU 6.2 Allocate
- LU 6.2 Get_Conversation
- CPI-C Initialize_Conversation
- CPI-C Accept_Conversation.

For example, say that two programs call the LU 6.2 Allocate service from the same address space and both calls to Allocate receive error return codes. APPC/MVS assigns a *different* conversation ID to each call. Your TP can specify either of those conversation IDs on calls to `Error_Extract`, even though the conversations were never actually allocated.

If APPC/MVS is not active, your TP will not be able to establish a conversation. In this case, your TP receives decimal return code 20 (product-specific error) from the Allocate call. To verify that APPC/MVS is not active, call `Error_Extract`, which returns decimal return code 64 when APPC/MVS is not active. (To debug other product-specific errors, refer to [“Diagnosing Product-Specific Errors”](#) on page 114.)

Using `Error_Extract` for Synchronous and Asynchronous Calls

When designing your TP, you can choose to call some conversation services *synchronously* or *asynchronously*, depending on whether or not you want to process other instructions while the system processes a call. Your TP can call `Error_Extract` to return error information for both synchronous and asynchronous calls.

Calling `Error_Extract` for Synchronous Requests

When you select synchronous processing, the system must complete processing for the call before it moves on to process other instructions. To return error information for a synchronous call, your TP should call `Error_Extract` immediately after processing for the synchronous call is complete (which is indicated by the return code).

The following section shows an example of how to call `Error_Extract` for an synchronous request.

Example Call to `Error_Extract` (Synchronous)

[Figure 32 on page 111](#) shows how to call `Error_Extract` to return error information for a synchronous call to a conversation service:

```

/*****
/* Call the APPC/MVS LU6.2 Send_Data service. Specify a
/* Notify_type of None to request synchronous processing.
*****/

CALL ATBSEND(Conversation_id,
             Send_type,
             Send_length,
             Access_token,
             Buffer,
             Request_to_send_received,
             Notify_type,          /* Value of "None" */
             Return_code);

/*****
/* Check the return code that APPC/MVS returns to the caller.
/* If an error occurred on the call (indicated by an error return
/* code from APPC/MVS), call the Error_Extract service to
/* obtain a service reason code and error message; write the
/* error message to the output stream.
*****/

IF Return_code ^= atb_ok THEN
BEGIN
CALL ATBEES3(Conversation_id,
             Service_Name,
             Service_Reason_Code,
             Message_Text_Length,
             Message_Text,
             Error_Log_Product_Set_ID_Length,
             Error_Log_Product_Set_ID,
             Error_Log_Information_Length,
             Error_Log_Information,
             Reason_Code,
             Return_Code);
IF (Return_Code = 0) THEN
BEGIN
/*****
/* If the call to Error_Extract is successful,
/* write the message text returned by Error_Extract
/* to the output stream. In this example, only
/* messages with a length of 126 characters or less
/* are displayed (126 is the maximum message length
/* that DISPLAY can handle.) You might want to
/* display more of the message text with multiple
/* DISPLAY statements.
*****/
IF Message_Text_Length <= 126 THEN
DISPLAY (Message_Text);

/*****
/* If the partner TP provided a product set ID,
/* write it to the output stream. In this example,
/* we display only the software product name from
/* the subvector that contains the product set ID.
/* Your TP can extract parts of the product
/* set ID as desired. See 'Sending a Product Set ID
/* to a Partner System' in this section for infor-
/* mation about how to extract parts of the ID.
*****/
IF Error_Log_Product_Set_ID_Length > 0 THEN
CALL Extract_Software_Product_Name (Error_Log_Product_set_ID,
                                   Product_name_length,
                                   Product_name);

/*****
/* Write the software product name to the output
/* stream. This example program displays only
/* product names with a length of 126 characters or
/* less. Your TP can use multiple DISPLAY
/* statements to display product names with more
/* than 126 characters.
*****/
IF Product_name_length <= 126 THEN
DISPLAY (Product_name);

/*****
/* If the partner TP or system provided error log
/* data, write it to the output stream. This example
/* displays only product names with 126 characters
/* or less. Your TP can use multiple
/* DISPLAY statements to display product names with
/* more than 126 characters.
*****/
IF Error_Log_Information_Length > 0 THEN
IF Error_Log_Information_Length <= 126 THEN
DISPLAY (Error_Log_Information);

END;
ELSE
DISPLAY ('APPC/MVS Error Extract Service failed');
END;

```

Figure 32. Example Use of Error_Extract Service, Synchronous

Calling Error_Extract for Asynchronous Requests

To return error information for an asynchronous call to a conversation service, your TP should call Error_Extract immediately after:

- The TP receives an error return code from APPC/MVS, or
- The system posts the event control block (ECB) that indicates the completion of the service.

Your TP should not call any other APPC/MVS services for the same conversation until the ECB is posted (indicating completion of the asynchronous service).

The following section shows an example of how to call `Error_Extract` for an asynchronous request. See [“Using Asynchronous Services” on page 52](#) for detailed information about asynchronous processing for conversation callable services.

Example Call to `Error_Extract` (Asynchronous)

Figure 33 on page 112 shows how to call `Error_Extract` to return error information for an asynchronous call to a conversation service:

```

/*****
/* Call the APPC/MVS LU6.2 Allocate service. Specify a
/* Notify_type of ECB to request asynchronous processing.
*****/

CALL ATBALLC(Conversation_type,
              Sym_dest_name,
              Partner_lu_name,
              Mode_name,
              TP_name_length,
              TP_name,
              Return_control,
              Sync_level,
              Security_Type,
              User_ID,
              Password,
              Profile,
              User_token,
              Conversation_ID,
              Notify_type,      /* Specifies a value of ECB */
              TP_ID,
              Return_code);

/*****
/* Check the return code that APPC/MVS returns to the caller.
/*
/* * If an error occurred on the call (indicated by an error
/* return code from APPC/MVS), call the
/* Error_Extract service to obtain a reason code and error
/* message, and write the error message to the output stream.
/*
/* * If no errors occurred on the call (indicated by an error
/* return code from APPC/MVS, call a procedure that contains
/* assembler code to wait on the ECB. If an error occurs
/* while processing the service, call Error_Extract.
*****/

IF Return_code ^= atb_ok THEN
  BEGIN
    CALL Report_error (Conversation_ID) ;
  END;
ELSE
  BEGIN
    CALL Wait_processing (Notify_ECB);
    IF Notify_ECB.completion_code > 0 THEN
      CALL Report_error (Conversation_ID) ;
    END;
  RETURN;

```

Figure 33. Example Use of `Error_Extract` Service, Asynchronous (figure continued)


```

/*****
/* Call procedure Report_Error to report an error in the call */
/* to the Allocate service. If the call to Report_Error is */
/* successful, it writes the following to the output stream: */
/* * The name of the callable service in error */
/* * The error message from Error_Extract */
/* * Error log data, if it is available */
*****/

Report_Error: Procedure (Conv_id);
BEGIN;
    CALL ATBEES3(Conversation_id,
                 Service_Name,
                 Service_Reason_Code,
                 Message_Text_Length,
                 Message_Text,
                 Error_Log_Product_Set_ID_Length,
                 Error_Log_Product_Set_ID,
                 Error_Log_Information_Length,
                 Error_Log_Information,
                 Reason_Code,
                 Return_Code);
    IF Return_code = 0 THEN
        BEGIN
            DISPLAY (Service_Name) ;
            /*****
            /* If the call to Error_Extract is successful, */
            /* write the message text returned by Error_Extract */
            /* to the output stream. In this example, only */
            /* messages with a length of 126 characters or less */
            /* are displayed (126 is the maximum message length */
            /* that DISPLAY can handle.) You might want to */
            /* display more of the message text with multiple */
            /* DISPLAY statements. */
            *****/
            IF Message_Text_Length <= 126 THEN
                DISPLAY (Message_Text) ;
                /*****
                /* If the partner TP provided a product set ID, */
                /* write it to the output stream. In this example, */
                /* we display only the software product name from */
                /* the subvector that contains the product set ID. */
                /* Your TP can extract parts of the product */
                /* set ID as desired. */
                *****/
                IF Error_Log_product_set_ID_length > 0 THEN
                    CALL Extract_Software_Product_Name (Error_log_product_set_ID,
                                                         Product_name_length,
                                                         Product_name);
                    /*****
                    /* Write the software product name to the output */
                    /* stream. This example program displays up */
                    /* to 126 characters in the name (it is the maximum */
                    /* length that DISPLAY can handle). */
                    /* Your TP might want to display more */
                    /* characters with multiple DISPLAY statements. */
                    *****/
                    IF Product_name_length <= 126 THEN
                        DISPLAY (Product_name) ;
                        /*****
                        /* If the partner TP or system provided error log */
                        /* data, write it to the output stream. This example */
                        /* displays up to 126 characters of log data (it is */
                        /* the maximum length that DISPLAY can handle). */
                        /* Your TP might want to display more */
                        /* characters with multiple DISPLAY statements. */
                        *****/
                        IF Error_Log_Information_Length > 0 THEN
                            IF Error_Log_Information_Length <= 126 THEN
                                DISPLAY (Error_Log_Information) ;
                            END;
                        ELSE
                            DISPLAY ('APPC/MVS Error Extract Service failed') ;
                        END;
                    /* End of Report_Error procedure */
                END;
            END;
        END;
    END;

```

Figure 34. Example Use of Error_Extract Service, Asynchronous

Diagnosing Product-Specific Errors

If your TP receives a decimal return code 20 from a call, either:

- APPC/MVS is not active, or
- A product-specific error occurred.

This return code often has an accompanying symptom record in the logrec data set. If APPC/MVS is not active, however, a symptom record is not produced. In this case, your TP can call the Error_Extract service, which returns decimal code 64 when APPC/MVS is not active. No further diagnostic information is available when APPC/MVS is not active.

When a symptom record is recorded in the logrec data set, section 3 of the record contains the primary symptom string for the product-specific errors:

Symptom	Description
-----	-----
PIDS/5752SCACB	Product identifier
RIDS/ATBxxxxx	CSECT name
RIDS/ATBxxxxx#L	Load module name
LVLS/ddd	Product level
PCSS/ATBxxxx or CMxxxx	Statement that caused the error
PRCS/dddddddd	Return code returned to the caller
FLDS/REASON VALU/Hddddddddd	Unique reason code identifying the product-specific error

Section 5 of the symptom record contains the following information for the product-specific error:

- The job or user name (in EBCDIC) for the home address space of the caller
- An EBCDIC description of the error (up to 80 characters)

Symptom FLDS/REASON VALU/Hddddddddd in section 3 of the symptom record contains one of the reason codes described in [Table 12 on page 114](#):

Table 12. Reason Codes for Product-Specific Errors		
Reason Code	Message Text	Explanation
00000001	APPC SERVICE REQUESTED WHILE SYSTEM LOCK HELD.	A user requested an APPC/MVS service while a system lock was held.
00000002	UNRECOGNIZED REQUEST.	The system request from the caller is not one of the APPC CPI-C or APPC LU 6.2 calls. The program might be using an incorrect level of the stub routine.

Table 12. Reason Codes for Product-Specific Errors (continued)

Reason Code	Message Text	Explanation
All other reason code values	Matches associated ATB8xxxxI message text.	<p>All product-specific error reason codes, except X'00000001' and X'00000002', are associated with a specific ATB8xxxxI message. You can use either the API trace facility or the Error_Extract service to obtain the message text, or:</p> <ol style="list-style-type: none"> 1. Convert the product-specific error reason code to decimal 2. Use the decimal reason code value to find the associated ATB8xxxxI message through the table in “Summary of Error_Extract Reason Codes” on page 313. 3. Refer to the appropriate ATB8xxxxI message description in “Error_Extract (ATB8) Messages” on page 343 for an explanation, system action, and suggested response for the error.

Part 3. Reference

Chapter 7. Invocation Details for APPC/MVS Callable Services

The APPC/MVS interfaces are written as callable services. They are grouped according to the class of programs that are likely to use the interfaces.

TP Conversation Services

Provide access to all APPC conversation functions. This set of services, used by transaction programs, has equivalent VTAM LU 6.2 verbs and CPI Communication calls.

Advanced TP Services

Provide access to more advanced transaction program interfaces, such as those used by TPs with a schedule type of Multi-trans.

Allocate Queue Services

Provide access to LU 6.2 server functions, which allow you to direct inbound allocate requests to server address spaces. These services have no equivalent SNA LU 6.2 services or CPI Communications routines. The APPC/MVS allocate queue services are documented in [*z/OS MVS Programming: Writing Servers for APPC/MVS*](#).

System Services

Provide access to system services not normally used by transaction programs, but used by other MVS components, management subsystems, and transaction schedulers. The APPC/MVS System services are documented in [*z/OS MVS System Messages, Vol 3 \(ASB-BPX\)*](#).

APPC/MVS Program Environment

Any MVS program that invokes APPC/MVS services, or is attached by an APPC/MVS LU in response to an inbound request, must be running in the following environment when it invokes APPC/MVS services:

Authorization:

Supervisor state or problem state, any PSW key

Dispatchable unit mode:

Task or SRB mode

Cross memory mode:

Any PASN, any HASN, any SASN

AMODE:

31-bit

ASC mode:

Primary or access register (AR)

Interrupt Status:

Enabled for I/O and external interrupts

Locks:

No locks held

Control parameters:

All parameters must be addressable by the caller and in the primary address space, except for the *buffer* parameter of ATBSEND, ATBRCVI, ATBRCVW, and ATBEXAI, which may reside in another address space or a data space.

Certain processing options of the APPC/MVS services require the callers to be in supervisor state or in PSW key 0-7. Those requirements are included with the reference information for each service.

High-Level Language Compilers

Table 13 on page 120 shows a partial list of high-level language compilers that support APPC/MVS calls on MVS/ESA. Calls can be made with other compiler levels and other compiler products that meet the preceding requirements and linkage conventions. Note that the requirement for 31-bit addressing may limit some language functions that you can use.

Table 13. Some High-Level Language Compilers for APPC/MVS Calls	
Language	Compiler
C	C/C++ for OS/390
COBOL	COBOL for OS/390 & VM Version 2
FORTRAN	VS FORTRAN Compiler Version 2, Release 6 0
PL/I	PL/I for MVS & VM Version 1, Release 1
RPG	RPG/370 Version 1, Release 1.0

Syntax and Linkage Conventions for the Callable Services

All APPC/MVS callable services have a general calling syntax as follows:

```
CALL routine_name      (parameters,return_code)
```

Some specific calling formats for languages that can invoke the APPC/MVS callable services are:

COBOL

```
CALL "routine_name" USING parm1,parm2,...return_code
```

FORTRAN

```
CALL routine_name (parm1,parm2,...return_code)
```

C

```
routine_name (parm1,parm2,...return_code)
```

PL/I

```
CALL routine_name (parm1,parm2,...return_code)
```

REXX

```
ADDRESS LU62 "routine_name parm1 parm2...return_code"
```

or

```
ADDRESS LINKPGM "routine_name parm1 parm2...return_code"
```

For REXX, enclose the routine name and all parameters within one pair of single or double quotes. Parameters must be initialized to appropriate values. The host command environment resolves the parameter values. For more information, see [z/OS TSO/E REXX Reference](#).

Assembler Call macro

```
CALL routine_name,(parm1,parm2,...return_code),VL
```

Callers must also use the following linkage conventions for all APPC/MVS services:

- Register 1 must contain the address of a parameter list, which is a list of consecutive words, each containing the address of a parameter to be passed. The last word in this list must have a 1 in the high-order (sign) bit.
- Register 13 must contain the address of an 18-word save area.
- Register 14 must contain the return address.
- Register 15 must contain the entry point address of the service being called.
- If the caller is running in AR ASC mode, access registers 1, 13, 14, and 15 must all be set to zero.

On return from the service, general and access registers 2 through 14 are restored (registers 0, 1 and 15 are not restored).

Any high-level language that generates this type of interface may be used to invoke APPC/MVS callable services.

Parameter Description for Callable Services

All the parameters of the APPC/MVS callable services are required positional parameters. When you invoke a service, you must specify all the parameters in the order listed. APPC/MVS checks all parameters for valid values, regardless of whether the parameters are used in call processing. Even though a language may allow parameters to be omitted, APPC/MVS services do not.

Some parameters do not require values and allow you to substitute zeros or a string of blanks for the parameter. For example, if you do not specify a symbolic destination on the Allocate call, you must set the `Sym_dest_name` parameter to eight blanks. The descriptions of the parameters identify those which can be replaced by blanks or zeros, and when to do so.

In the descriptions of services in this document, each parameter is described as supplied or returned.

Supplied means that you supply a value for the parameter in the call.

Returned means that the service returns a value in the named parameter when the call is finished (for example, `return_code`).

Each parameter is also described in terms of its data type, character set, and length.

Data type is either integer, character string, or structure.

Character set applies only to parameters whose values are character strings and governs the values allowed for that parameter. Possible character sets are:

- No restriction

There is no restriction on the byte values contained in the character string.

- Type A EBCDIC

The string may contain only uppercase alphabetics, numerics, and national characters (@, \$, #), and must begin with an alphabetic or national character. Use of @, \$, and # is discouraged because those characters display differently on different national code pages.

- 01134

The string may contain uppercase alphabetics or numerics, with no restriction on the first character.

- 00640

The string may contain upper- or lowercase alphabetics, numerics, or any of 19 special characters with no restriction on the first character. This set is consistent with the 00640 character set except that APPC/MVS does not allow blanks in 00640 character strings.

For more detailed information about the characters in each character set, see [Appendix A, “Character Sets,”](#) on page 387.

Length depends on the data type of the parameter.

- For an integer item, the length indicates the size of the field in bits or bytes.
- For a character string parameter, the length value indicates the number of characters that may be contained in a character type parameter. The length may specify a single number or a minimum and maximum number.
- For a structure parameter, the length value indicates the size of the structure in bytes, or a minimum and maximum size if the size of the structure is variable.

Required Modules

The two methods described here can be used to access the APPC/MVS system services.

- One or more of the following modules from SYS1.CSSLIB must be link-edited with any program that issues APPC/MVS services:

ATBPBI

With programs that issue CPI Communications calls or TP conversation services

ATBATP

With programs that issue APPC/MVS advanced TP services

ATBCTS

With programs that issue APPC/MVS allocate queue services, or the Reject_Conversation or Set_Conversation_Accounting_Information services.

ATBCSS

With programs that issue APPC/MVS system services.

If the load modules are to be executed on a level of APPC/MVS other than the one on which the link-edit is performed, the link-edit should be run using copies of the SYS1.CSSLIB modules from the system on which the load modules will be executed.

After new releases of MVS are installed or maintenance is applied which affects this interface, these modules and any load modules containing copies of them must be link-edited with the APPC/MVS programs again. Therefore, with any APPC/MVS applications that you write, provide a post-install job to link-edit the modules again with the appropriate programs.

- A program can issue the MVS LOAD macro for the APPC/MVS service to obtain its entry point address, then use that address to call the APPC/MVS service.

Additional language-specific statements may be necessary so that language compilers can provide the proper assembler interface. Other programming notation, such as variable declarations, are also language-dependent.

Versions of Callable Services

New APPC/MVS callable services have a version number as the last character of the call name (for example, ATBIDN1). That number corresponds to the version of APPC/MVS in which the call was introduced.

To determine which calls are valid on a system, you can obtain the current APPC/MVS version number from the APPC/MVS Version service described in [“Version_Service” on page 287](#). On any system, valid APPC/MVS callable services include the following:

- Calls with no version number
- Calls with a version number less than or equal to the current APPC/MVS version number obtained from the Version service.

You may only call services that were introduced with the version number obtained from the version service. For example, calls to ATBCMCTU and ATBCUC1 are both valid when the current APPC/MVS version number is 1 or higher, but calls to ATBxxx2 would be valid only when the current APPC/MVS version number is 2 or higher.

Interface Definition Files (IDFs) for LU6.2 and APPC/MVS Services

APPC/MVS provides IDFs (also called pseudonym files or headers) which define variables and values for parameters of APPC/MVS services. IDFs are available for different languages, and can be included or copied from a central library into programs that invoke APPC/MVS callable services. The following IDFs are available on MVS:

For a list of IDFs for CPI-C calls, see [“Interface Definition Files \(IDFs\) for CPI-C Calls” on page 38](#).

APPC/MVS provides the following IDFs for APPC/MVS conversation calls:

<i>Table 14. IDFs for APPC/MVS Conversation Calls</i>	
Language	In member:
Assembler	ATBASASM in SYS1.MACLIB
Assembler	ATBSERV in SYS1.MACLIB
C	ATBPBC in SYS1.SIEAHDR.H
Note: ATBPBC is also shipped in the z/OS UNIX System Services HFS directory /usr/include.	
COBOL	ATBPBCOB in SYS1.SIEAHDR.H
FORTRAN	ATBPBFOR in SYS1.SIEAHDR.H
PL/I	ATBPBPLI in SYS1.SIEAHDR.H
REXX	ATBPBREX in SYS1.SIEAHDR.H

For APPC/MVS allocate queue services, and for the Reject_Conversation and Set_Conversation_Accounting_Information services:

<i>Table 15. IDFs for APPC/MVS Allocate Queue Services</i>	
Language	In member:
Assembler	ATBCTASM in SYS1.MACLIB
C	ATBCTC in SYS1.SIEAHDR.H
Note: ATBCTC is also shipped in the z/OS UNIX System Services HFS directory /usr/include.	
COBOL	ATBCTCOB in SYS1.SIEAHDR.H
FORTRAN	ATBCTFOR in SYS1.SIEAHDR.H
PL/I	ATBCTPLI in SYS1.SIEAHDR.H
REXX	ATBCTREX in SYS1.SIEAHDR.H

Chapter 8. APPC/MVS TP Conversation Callable Services

The APPC/MVS TP conversation callable services are based upon existing SNA LU 6.2 verbs and CPI Communications calls. These services are intended for use in conversations by transaction programs and APPC/MVS servers.

Programs that use APPC/MVS TP conversation services must use one of the two methods described “Required Modules” on page 122 to access the APPC/MVS system services.

The following table lists the TP conversation callable services that have more than one associated call name. This chapter describes the current versions of the calls, which are the preferred programming interfaces for these services. The previous versions are described in [Appendix E, “Previous Versions of APPC/MVS Callable Services,”](#) on page 425.

<i>Table 16. TP Conversation Callable Services with Multiple Call Names</i>			
Service Name	Previous Call Name	Current Call Name	Reference for Current Call
Allocate	ATBALLC and ATBAL2	ATBALC5	“ATBALC5 - Allocate (For OS/390 Release 8 through z/OS V1R6)” on page 448
Get_TP_Properties	ATBGTP	ATBGTP4	“Get_TP_Properties” on page 175

Allocate

Equivalent to:

- LU 6.2 (MC_)Allocate
- CPI Initialize_Conv (CMINIT) and Allocate (CMALLC)

Allocates a session between the local LU and a partner LU, and on that session allocates a basic or mapped conversation between the local program and a partner program. A conversation ID is assigned to the conversation. Call this service before other calls that refer to the conversation.

If the program that issues the allocate call was not started by APPC/MVS in response to an inbound allocate call, and is not associated with an alternative transaction scheduler, the outbound allocate call and ensuing conversation flow through the base LU for the APPC/MVS transaction scheduler. If, in such a case, there is no base LU defined for the APPC/MVS transaction scheduler, APPC/MVS uses the system base LU. If there is no system base LU, APPC/MVS rejects the allocate request. For more information about base LUs and their definition, see [z/OS MVS Planning: APPC/MVS Management](#).

Requirements

Authorization:	<p>Supervisor state or problem state, any PSW key, with the following exceptions:</p> <ul style="list-style-type: none"> • When the TP_name specified is an SNA TP name beginning with X'06', the caller must run either in supervisor state, or with PSW key 0-7. • When the TP_id specified is a value other than binary zeros, the caller must run either in supervisor state, or with PSW key 0-7.
-----------------------	--

Dispatchable unit mode:	Task or SRB mode, with the following exception: task mode only for callers that issue Allocate for a conversation with a synchronization level of Syncpt.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBALC6(Conversation_type,
             Sym_dest_name,
             Partner_LU_name,
             Mode_name,
             TP_name_length,
             TP_name,
             Return_control,
             Sync_level,
             Security_type,
             User_ID,
             Password,
             Profile,
             User_Token,
             Conversation_ID,
             Notify_type,
             TP_ID,
             Local_LU_name,
             Timeout_value_minutes,
             Timeout_value_seconds,
             Return_code
            );
```

Figure 35. ATBALC6 - LU 6.2 Allocate

Parameters

Conversation_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the service is invoked.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

Specifies that in this conversation, the TPs will format their data into separate records, with record length and data specified, before sending it.

1

Mapped_conversation

Specifies that in this conversation, the TPs will rely on APPC to format the data that the TPs send.

Sym_dest_name

Supplied parameter

- Type: Character string
- Char Set: 01134
- Length: 8 bytes

Specifies a symbolic name representing the partner LU, the partner TP_name, and the mode name for the session on which the conversation is to be carried. The symbolic destination name must match that of an entry in the side information data set. The appropriate entry in the side information is retrieved and used to initialize the characteristics for the conversation.

If you specify a symbolic destination name, the partner LU name, mode name, and TP name are obtained from the side information. If you also specify values for the Partner_LU_name, Mode_name, or TP_name parameters on the Allocate service, these values override any obtained from the side information.

The symbolic destination name in this field can be from 1 to 8 characters long, with characters from character set 01134. If the symbolic destination name is shorter than eight characters, it must be left-justified in the variable field, and padded on the right with blanks. To not specify a symbolic destination name, set the sym_dest_name parameter value to 8 blanks and provide values for the Partner_LU_name, Mode_name, and TP_name parameters.

Partner_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes (must be padded with blanks if less than 17 bytes)

Partner_LU_name specifies the name of the LU at which the partner program is located.

The Partner_LU_name is any name by which the local LU knows the partner LU for the purposes of allocating a conversation. The local LU transforms this locally known LU name to an LU name used by the network.

The Partner_LU_name can be one of the following:

- LU name only (1-8 byte Type A character string).

This string represents the network LU name, which, if unique within the network and interconnected networks, is sufficient for most TP communications.

IBM recommends, however, that you specify either a symbolic destination name (in the Sym_dest_name parameter), or the network-qualified LU name, if known. While the network LU name might be unique currently, it might not remain so if the installation increases the number of networks in use. Specifying a symbolic destination name or network-qualified LU name can minimize the need for future network definitions and program changes.

- A VTAM generic resource name.

If the partner LU is a member of a generic resource group, you may specify the 1- to 8-byte generic resource name of the group.

- Combined network_ID and network LU name (two 1-8 byte Type A character strings, concatenated by a period: *network_ID.network_LU_name*).

This format is known as a **network-qualified LU name**; each LU in the network and all interconnected networks can be uniquely identified by its network-qualified LU name. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name. If the local LU is not enabled to support network-qualified names, APPC/MVS passes only the network-LU-name portion to VTAM, which might cause an error if the network LU name is not unique across networks.

- A value of 17 blanks:

If you specify a symbolic destination name in Sym_dest_name parameter, set Partner_LU_name to blanks to use the partner LU name from the side information.

If you do not specify a symbolic destination name, then use a blank Partner_LU_name to indicate that the partner program is located at the same LU as the local program (LU=OWN). If the local LU is defined as a member of a VTAM generic resource group, APPC/MVS uses the generic resource name for Partner_LU_name.

Mode_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes (must be padded with blanks if less than 8 bytes)

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

The mode name value of "SNASVCMG" is reserved for use by APPC/MVS. If a mode name of "SNASVCMG" is specified on the Allocate service, the request is rejected with a return code of parameter_error.

If you specify a symbolic destination name in the sym_dest_name parameter, set mode_name to blanks to obtain the mode_name from the side information.

If you do not specify a sym_dest_name and do not specify a mode name, APPC/MVS uses the default mode name "ATB#MODE".

TP_name_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

TP_name_length specifies the length of data contained in the TP_name parameter.

If you specify a symbolic destination name in the sym_dest_name parameter, set TP_name_length to 0 to use the partner TP name from the side information.

TP_name

Supplied parameter

- Type: Character string
- Char Set: 006409 (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

TP_name specifies the name of the partner program to be connected at the other end of the conversation.

If you specify a symbolic destination name in the sym_dest_name parameter and set the TP_name_length parameter to zero, the TP name is obtained from the side information file.

TP_name can specify the name of any SNA service transaction program except for one whose first character is X'06'; see the authorization requirements in ["Requirements" on page 448](#) for more information about this exception. The names of SNA service transaction programs can contain blank characters. For a list of SNA service transaction programs, see *SNA Transaction Programmer's Reference Document for LU 6.2*.

If the partner TP is to be protected by a RACF security profile in the APPCTP class, the TP_name must consist of Type A characters only.

Return_control

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_control specifies when the local LU is to return control to the local program, in relation to the allocation of a session for the conversation.

Valid values for this parameter are:

Value

Meaning

0

When_session_allocated

Specifies to allocate a session for the conversation before returning control to the program. An error in allocating a session is reported on this call.

1

Immediate

Specifies to allocate a session for the conversation if a session is immediately available, and return control to the program with a return code indicating whether a session is allocated. An error in allocating a session that is immediately available is reported on this call.

100

When_conwinner_allocated

Specifies to allocate a session in which the local LU is the contention winner, before returning control to the program. As contention winner, the LU avoids having to compete with the partner LU to establish the session, thus potentially saving network traffic. An error in allocating a contention winner session for the conversation is reported on this call.

Sync_level

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not call any services and will not recognize any returned parameters relating to confirmation.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs can call services and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Security_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Security_type specifies the type of access security information that the partner LU uses to verify the identity of the end-user and validate access to the partner program and its resources. Before you choose a value for Security_type, make sure you know the level of security that the partner LU accepts. If the Security_type parameter and the security accepted by the partner LU do not match, the system may not return expected security information. See [Requesting that VTAM Verify Partner LUs and Defining Conversation Security Levels that Sessions Allow in z/OS MVS Planning: APPC/MVS Management](#).

Valid values for this parameter are:

Value
Meaning
100

Security_none

Specifies to omit access security information on this allocation request.

101

Security_same

Specifies to use the same user ID that is associated with the current program the Allocate service is issued from. The password (if present) is not used; instead, the user ID is indicated as being already verified. If the allocation request that initiated execution of the local program contained no security information, security information is omitted on this allocation request. APPC can retrieve the security information from a number of different places. If the user is authorized and the user specifies a valid User-Token parameter, APPC will use this to obtain the appropriate security information (a user ID and possible profile name). If this is not specified, APPC will send the user ID associated with the current application context environment, if this is available. Otherwise, APPC will send the user ID and possible profile name that is associated with the current executing task, or if unavailable, from the current address space.

102

Security_pgm

Specifies to use the access security information that the local program provides on the call. The local program provides the information by means of the User_ID, Password, and Profile parameters. These values are passed exactly as specified, without folding to uppercase.

Normally, User_ID and Password are required parameters for this Security_type. However, the User_ID parameter can be specified without the Password parameter if, on the local system, the user ID of the issuing address space has been granted surrogate authorization for the specified User_ID. In RACF terms, this requires READ access to the ATBALLC.userid profile (or a generic profile) in the SURROGAT class, where *userid* is the value specified on the User_ID parameter. If surrogate authorization is granted, the user ID specified on the call will be sent and will be indicated as being already verified. For general information on surrogate user IDs, see [z/OS Security Server RACF Security Administrator's Guide](#). For specific information about ATBALLC.userid profiles, see [z/OS MVS Planning: APPC/MVS Management](#).

Note: If surrogate authorization is used, the specified User_ID must be a valid MVS user ID. For example, it cannot be longer than 8 characters.

User_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)

- Length: 10 bytes

Specifies the user ID. The partner LU uses this value and the password to verify the identity of the end user that initiated the allocation request. The partner LU may use this value for auditing and accounting purposes, and, together with the security profile (if present), to determine which partner programs the local program can access.

When the partner LU is on MVS with RACF protection, the user ID must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Password

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (must be left-justified and padded with blanks if less than 10 bytes)

Specifies the password. The partner LU uses this value and the user ID to verify the identity of the end user that made the allocation request. When the partner LU is on MVS with RACF protection, the password must be 1-8 alphanumeric characters padded with blanks.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Profile

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Profile specifies additional security information that may be used to determine what partner programs the local program may access, and which resources the local program may access. When the partner LU is on MVS with RACF protection, APPC/MVS treats the profile name as a RACF group name for verifying access to partner programs. The profile name must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

User_Token

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 1-255 bytes

User_Token specifies the RACF UTOKEN which identifies the user requesting the Allocate. Only programs running in supervisor state or PSW key 0-7 can specify a User_Token. To not specify a User_Token, pass a field whose first byte contains a hexadecimal zero (X'00').

If a RACF UTOKEN is supplied, APPC/MVS uses it to obtain the appropriate security information only when you specify a Security_Type of Security_Same. In that case, APPC/MVS obtains the user ID and RACF group name from the UTOKEN. This parameter will not be consulted if Security_Type is Security_None or Security_Pgm.

Conversation_id

Returned parameter

- Type: Character string
- Char Set: No restriction

- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Note: As of MVS/ESA SP 4.2.2, unauthorized callers can specify a Notify_type of ECB on calls to Allocate. With MVS/ESA SP 4.2, unauthorized callers cannot specify a Notify_type of ECB.

TP_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Allows authorized TPs to designate the transaction program instance with which this conversation should be associated. (See “Requirements” on page 448 for more information about specific authorization requirements.) Unauthorized TPs must set this parameter to binary zeros, which causes the TP_ID assignment to occur automatically and transparently to the transaction program.

Advanced TPs that run in supervisor state or PSW key 0-7 can select the TP_ID assigned. See the Define_Local_TP callable service description in *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for information on how to create a new TP_ID.

Local_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Local_LU_name specifies the name of the local LU from which the caller's allocate request is to originate. The ability to specify the local LU name allows the caller to associate its outbound

conversations with particular LUs. You cannot specify a VTAM generic resource name for the local LU name.

The caller's address space must have access to the named LU. Otherwise, a parameter_error return code is returned. Use [Table 17 on page 133](#) to determine whether you can specify a particular local LU.

<i>Table 17. Local LUs for Which an Address Space Can Allocate</i>						
		LU Specified				
		<i>System Base LU, NOSCHED¹</i>	<i>System Base LU, ASCH¹</i>	<i>NOSCHED LU</i>	<i>ASCH LU</i>	<i>Scheduler 2 LU</i>
Address Space Doing Allocate	<i>From an Address Space Not Connected to a Scheduler</i>	OK	OK	OK	NO ²	NO ²
	<i>From an Address Space Connected to ASCH</i>	OK	OK	OK	OK	NO ²
	<i>From an Address Space Connected to Scheduler 2</i>	OK	NO ²	OK	NO ²	OK
	<i>From an Address Space Not Connected to a Scheduler with Prohibit Default LU Specified⁴</i>	NO ³	NO ³	NO ³	NO ³	NO ³
Note: <ol style="list-style-type: none"> Columns 1 and 2 are mutually exclusive. The system returns a Parameter_error return code to the caller. If the specified LU is not defined, the system also returns a Product_specific_error return code to the caller. The system returns a Product_specific_error return code to the caller. For information about how to prohibit the use of a default LU for an address space, see the description of the Set_AS_Attributes service in <i>z/OS MVS System Messages, Vol 3 (ASB-BPX)</i>. 						

If the caller sets local_LU_name to blanks, the system uses the following hierarchy to select an LU for the conversation:

1. The LU associated with the transaction program
2. If no LU is associated with the TP, the system uses the base LU for the transaction scheduler associated with the caller's address space.
3. If no transaction scheduler is associated with this address space, the system uses the system base LU, which is either:

- An LU defined with the NOSCHED and BASE parameters, or
- If a base NOSCHED LU is not defined, the LU defined as the base LU for the APPC/MVS transaction scheduler.

4. If no system base LU is defined, the system rejects the Allocate call.

For more information about base LUs and their definitions, see [z/OS MVS Planning: APPC/MVS Management](#).

Table 18 on page 134 shows which LU is used by default.

<i>Table 18. Default Local LUs Used If None Are Specified</i>							
Program Calling Allocate Service	Base LUs exist						
	<i>nosched</i>	<i>ASCH</i>	<i>Sched 2</i>	<i>nosched, ASCH</i>	<i>nosched, Sched 2</i>	<i>ASCH, Sched 2</i>	<i>nosched, ASCH, Sched 2</i>
<i>From an Address Space Not Connected to a Scheduler</i>	nosched	asch	NO ¹	nosched	nosched	asch	nosched
<i>From an Address Space Not Connected to a Scheduler but with prohibit default LU specified</i>	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹
<i>From an Address Space Connected to ASCH</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an Address Space Connected to ASCH and with prohibit default LU specified</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an Address Space Connected to Scheduler 2</i>	nosched	NO ¹	Sched 2	nosched	Sched 2	Sched 2	Sched 2
<i>From an Address Space Connected to Scheduler 2 and with prohibit default LU specified</i>	NO ¹	NO ¹	Sched 2	NO ¹	Sched 2	Sched 2	Sched 2
Note:							
1. A Product_Specific_Error return code is returned if no base LU exists.							

Timeout_Value_Minutes

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

- Value range: 0-1440 (decimal)

Sets a time limit in minutes that an allocate call and subsequent APPC/MVS TP conversation calls will wait for VTAM APPCCMD requests to complete. For more information, see [“Setting a Timeout Value for Potential Network Delays”](#) on page 55.

If the time limit is reached before the VTAM APPCCMD request completes and returns control to APPC/MVS, the conversation will be terminated by APPC/MVS and the caller of the conversation callable service will regain control. If the conversation call was issued with a Notify_Type=ECB (asynchronous processing), the specified ECB will be posted when the time limit is reached.

To alter the timeout_value set on the Allocate service, use the Set_Timeout_Value service.

The maximum supported value for Timeout_Value_Minutes is 1440 minutes (24 hours). A value for Timeout_Value_Minutes of 0 and any other positive integer (≤ 1440) is valid. The total APPC timeout value is obtained from this parameter and the Timeout_Value_Seconds parameter. The two values are combined together to give the complete amount of time APPC should wait before timing out the conversation. If both Timeout_Value_Minutes and Timeout_Value_Seconds are zero, the allocate call and any subsequent VTAM APPCCMD requests issued by subsequent APPC/MVS conversation callable services will not be timed. In this case, you can activate the time-out feature later by invoking the Set_Timeout_Value conversation callable service and specifying a non-zero Timeout_Value_Minutes and/or Timeout_Value_Seconds.

When either the Timeout_Value_Minutes or Timeout_Value_Seconds parameter is non-zero and a VTAM APPCCMD request issued during allocate processing does not complete within the time-out period, the conversation allocation will fail and APPC/MVS will return control to the application with a Product_Specific_Error return code.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Timeout_Value_Seconds

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Value range: 0-59 (decimal)

Sets a time limit in seconds that an allocate call and subsequent APPC/MVS TP conversation calls will wait for VTAM APPCCMD requests to complete. For more information, see [“Setting a Timeout Value for Potential Network Delays”](#) on page 55.

If the time limit is reached before the VTAM APPCCMD request completes and returns control to APPC/MVS, the conversation will be terminated by APPC/MVS and the caller of the conversation callable service will regain control. If the conversation call was issued with a Notify_Type=ECB (asynchronous processing), the specified ECB will be posted when the time limit is reached.

To alter the timeout_value set on the Allocate service, use the Set_Timeout_Value service.

The maximum supported value for Timeout_Value_Seconds is 59 seconds. A value for Timeout_Value_Seconds of 0 and any other positive integer (≤ 59) is valid. The total APPC timeout value is obtained from this parameter and the Timeout_Value_Minutes parameter. The two values are combined together to give the complete amount of time APPC should wait before timing out the conversation. If both Timeout_Value_Minutes and Timeout_Value_Seconds are zero, the allocate call and any subsequent VTAM APPCCMD requests issued by subsequent APPC/MVS conversation callable services will not be timed. In this case, you can activate the time-out feature later by invoking the Set_Timeout_Value conversation callable service and specifying a non-zero Timeout_Value_Minutes and/or Timeout_Value_Seconds.

When either the Timeout_Value_Minutes or Timeout_Value_Seconds parameter is non-zero and a VTAM APPCCMD request issued during allocate processing does not complete within the time-out period, the conversation allocation will fail and APPC/MVS will return control to the application with

a Product_Specific_Error return code. Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call because nothing is placed in the variables.

When APPC/MVS returns an error return code to Allocate, your TP:

- Can use the conversation ID returned on the Conversation_ID parameter as input to the Error_Extract service (which returns detailed information about error return codes)
- Should not examine any other returned parameter associated with the call because no values are placed in the parameters.

An allocation error resulting from the local LU's failure to obtain a session for the conversation is reported on this call. An allocation error resulting from the partner LU's rejection of the allocation request is reported on a subsequent call.

See [“Return Codes” on page 458](#) for descriptions of return codes that can be returned to a caller of Allocate.

Return Codes

If the Return_control parameter contains a value of When_session_allocated or When_conwinner_allocated, possible values of Return_code are:

Decimal Value

Meaning

0	OK
1	Allocate_failure_no_retry
2	Allocate_failure_retry
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check

If the Return_control parameter contains a value of Immediate, possible values of Return_code are:

Decimal Value

Meaning

0	OK
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check
28	Unsuccessful

The following table describes *all* of the possible return codes for Allocate:

<i>Table 19. Return Codes for the Allocate Service</i>	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
1	<p>Value: Allocate_failure_no_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> Virtual telecommunications access method (VTAM) could not establish a session with the partner LU. APPC/MVS could not establish a conversation. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p> <p>If the conversation is not LU=LOCAL, see <i>z/OS Communications Server: SNA Programmer's LU 6.2 Guide</i> for a description of the sense codes included in the message from Error_Extract. If the error persists, or if the conversation is LU=LOCAL, verify that the name specified on the Local_LU_name parameter is correct. If the name is correct, contact the system programmer.</p> <p>System Programmer Response: At the request of the application programmer, ensure that the local LU is defined correctly in the VTAM application (APPL) statement in SYS1.VTAMLST.</p>

<i>Table 19. Return Codes for the Allocate Service (continued)</i>	
Return Code	Value, Meaning, and Action
2	<p>Value: Allocate_failure_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. The system cannot allocate the conversation because of a condition that might be temporary.</p> <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: Retry the allocate request.</p>
7	<p>Value: Sync_lvl_not_supported_lu</p> <p>Meaning: A TP submitted an Allocate request with a synchronization level that is not supported by the partner LU.</p> <p>System Action: The system returns this return code to the caller of the Allocate call.</p> <p>Application Programmer Response: Ensure that the partner LU supports the receipt of conversations with a synchronization level of syncpt.</p>
19	<p>Value: Parameter_error</p> <p>Meaning: A local TP called an APPC service. A parameter specified on the call is not valid. The error could be one of the following:</p> <ul style="list-style-type: none"> • The TP name is not 1 to 64 characters long. • Either the SYMDEST name or the TP name length were not specified. • SNASVCMG is specified as mode name. • X'0E' or X'0F' was used as the first character of a TP name. • X'06' was used as the first character of a TP name by a caller that was not running either in supervisor state or with PSW key 0-7. • An SNA service TP name is used with a mapped conversation verb. • The partner LU name was not valid. • The mode name was not valid. • The local LU name specified is either undefined or not allowed; for example, the TP might have specified a VTAM generic resource name, which is valid only for partner LU names. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 19. Return Codes for the Allocate Service (continued)

Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The local TP called an APPC service. One of the following errors occurred in one or more parameters specified on the call:</p> <ul style="list-style-type: none"> • An unauthorized caller passed a non-zero TP_ID. • For a Security_type of Security_pgm, both the user ID and password were not specified. • For a Security_type of Security_pgm, a user ID was specified with a blank password, or a password was specified with a blank user ID. • The SYMDEST name was not found in the side information. • The specified TP_ID is not associated with the address space. • An unauthorized caller specified a UTOKEN that was non-zero. • The specified local LU does not support protected conversations (conversations with a synchronization level of syncpt). • The specified Timeout_value is not valid <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: For a conversation with sync_level set to SYNCPT, the conversation's context (unit of work) is in the Backout-Required condition. New protected conversations cannot be allocated for a context in this condition.</p> <p>System Action: The conversation allocation request fails. A new conversation is not allocated.</p> <p>Application Programmer Response: Backout the current unit of recovery associated with the transaction program's context.</p>
28	<p>Value: Unsuccessful</p> <p>Meaning: The request specified an allocate_type of <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> • APPC/MVS could not establish a session with the partner LU • Virtual telecommunications access method (VTAM) could not establish a conversation. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

For more detailed information about these return codes, refer to [Appendix B, “Explanations of Return Codes for CPI Communications Services,” on page 391](#).

Restrictions

Transaction programs that call the Allocate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT

FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Confirm

Equivalent to:

- LU 6.2 (MC_)Confirm
- CPI Confirm (CMCFM)

Sends a confirmation request to the partner program and waits for a reply. This service allows the local and partner programs to synchronize their processing with one another. The LU flushes its send buffer as a function of this service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBCFM(  
    Conversation_id,  
    Request_to_send_received,  
    Notify_type,  
    Return_code  
);
```

Figure 36. ATBCFM - LU 6.2 Confirm

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Request_to_send_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Request_to_send_received specifies whether or not Request_to_send notification has been received.

Valid return values for this parameter are:

Value

Meaning

0

Request_to_send_not_received

Indicates that Request_to_send notification has not been received

1

Request_to_send_received

Indicates that the partner program has issued a Request_to_send, requesting the local program to enter Receive state.

If Return_code indicates Program_parameter_check or Program_state_check, a value is not returned in Request_to_send_received.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Note: Unauthorized callers can specify a Notify_Type of ECB on calls to Allocate.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where the system returns an error code of Program_parameter_check or Program_state_check, the program should not examine the Request_to_send_received parameter (the system does not return a value in the parameter). In all other cases, the system returns a value in the Request_to_send_received parameter (which the program can examine).

See the next section for descriptions of return codes that can be returned to a caller of Confirm.

Return Codes

Valid return code values for the Return_code parameter are:

Table 20. Return Codes for the Confirm Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 20. Return Codes for the Confirm Service (continued)

Return Code	Value, Meaning, and Action
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, call the Error_Extract service immediately after APPC/MVS returns this return code. See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for descriptions of the types of information that Error_Extract returns. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocated_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 20. Return Codes for the Confirm Service (continued)	
Return Code	Value, Meaning, and Action
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The local TP called an APPC service. One of the following errors occurred in one or more parameters specified on the call:</p> <ul style="list-style-type: none"> • An unauthorized caller specified a Notify_type of ECB. • The Sync_level field for the conversation was equal to sync_level_none. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: One of the following errors occurred:</p> <ul style="list-style-type: none"> • The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables. • The conversation was in send state, but the TP did not finish sending a logical record. • The conversation is not in Send or Send-pending state. • The Sync_level is set to syncpt, and the TP is in the Backout-required state. <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 20. Return Codes for the Confirm Service (continued)	
Return Code	Value, Meaning, and Action
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 20. Return Codes for the Confirm Service (continued)	
Return Code	Value, Meaning, and Action
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 20. Return Codes for the Confirm Service (continued)

Return Code	Value, Meaning, and Action
132	<p>Value: Deallocate_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

Transaction programs that call the Confirm service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Confirmed

Equivalent to:

- LU 6.2 (MC_)Confirmed

- CPI Confirmed

Sends a confirmation reply to the partner program. This service allows the local and partner programs to synchronize their processing with one another.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBCFMD(
    Conversation_id,
    Notify_type,
    Return_code
);
```

Figure 37. ATBCFMD - LU 6.2 Confirmed

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the next section for descriptions of return codes that can be returned to a caller of Confirmed.

Return Codes

Valid return code values for the Return_code parameter are:

Table 21. Return Codes for the Confirmed Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 21. Return Codes for the Confirmed Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: One of the following errors occurred:</p> <ul style="list-style-type: none"> The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables. The Sync_level is set to syncpt, and the TP is in the Backout-required state. <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Confirmed service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Deallocate

Equivalent to:

- LU 6.2 (MC_)Deallocate
- CPI Deallocate (CMDEAL)

Deallocates the specified conversation from the transaction program.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBDEAL(
    Conversation_id,
    Deallocate_type,
    Notify_type,
    Return_code
);
```

Figure 38. ATBDEAL - LU 6.2 Deallocate

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Deallocate_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Deallocate_type specifies the type of deallocation to be performed.

Valid values for this parameter are:

Value

Meaning

0

Deallocate_sync_level

Specifies to perform deallocation based on the synchronization level in effect for the conversation:

- If the synchronization level is None, execute the function of the Flush service and deallocate the conversation normally.
- If the synchronization level is Confirm, execute the function of the Confirm service; and if it is successful, deallocate the conversation normally.
- If the synchronization level is syncpt, the deallocation is deferred until the program issues a Commit call. Upon successful completion of the Commit call, the system deallocates the conversation.

If the Commit call is not successful or the program issues a Backout call, the system does not deallocate the conversation; instead, the conversation is restored to the state it was in at the previous synchronization point. (You can call Get_Attributes to determine the state of the conversation.) If there has not been a synchronization point since the allocation of this conversation, the conversation state for the TP that allocated this conversation is set to Send state.

1

Deallocate_flush

Specifies to execute the function of the Flush service and deallocate the conversation normally.

Note: You cannot specify Deallocate_flush for a conversation with a synchronization level of syncpt.

Deallocate

2

Deallocate_confirm

Specifies to execute the function of the Confirm service and if it is successful, deallocate the conversation normally.

Note: You cannot specify Deallocate_confirm for a conversation with a synchronization level of syncpt.

3

Deallocate_abend

Specifies to run the function of the Flush service and deallocate the conversation abnormally. Do not invoke deallocate_abend following a Post_on_Receipt. Transaction programs that issue Deallocate_abend after Post_on_Receipt might not regain control after issuing the deallocate.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the next section for descriptions of return codes that can be returned to a caller of Deallocate.

Return Codes

If the Deallocate_type parameter contained the value Deallocate_sync_level and the synchronization level of the conversation is none, or if the Deallocate_type parameter contained the value Deallocate_flush or Deallocate_abend, possible values for Return_code are:

Decimal Value

Meaning
0 OK
20 Product_specific_error
24 Program_parameter_check
25 Program_state_check

If the Deallocate_type parameter contained the value Deallocate_sync_level and the synchronization level of the conversation is confirm, or if the Deallocate_type parameter contained the value Deallocate_confirm, possible values for Return_code are:

Value

Meaning
0 OK
3 Conversation_type_mismatch
5 PIP_not_specified_correctly
6 Security_not_valid
8 Sync_lvl_not_supported_pgm
9 TPN_not_recognized
10 TP_not_available_no_retry
11 TP_not_available_retry
17 Deallocated_abend
20 Product_specific_error
22 Program_error_purging
24 Program_parameter_check
25 Program_state_check
26 Resource_failure_no_retry
27 Resource_failure_retry

Deallocate

30

Deallocated_abend_SVC

31

Deallocated_abend_timer

33

SVC_error_purging

If the Deallocate_type parameter contained the value Deallocate_sync_level and the synchronization level of the conversation is syncpt, possible values for Return_code are:

Value

Meaning

0

OK

20

Product_specific_error

24

Program_parameter_check

25

Program_state_check

The following table describes *all* of the possible return codes for Deallocate:

Table 22. Return Codes for the Deallocate Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>If the Deallocate_type parameter contained the value deallocate_sync_level and the synchronization level of the conversation is syncpoint, the system defers or does not perform the deallocation; see the Deallocate_type parameter description for more information.</p> <p>Application Programmer Response: None required.</p>
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>

Table 22. Return Codes for the Deallocate Service (continued)

Return Code	Value, Meaning, and Action
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 22. Return Codes for the Deallocate Service (continued)

Return Code	Value, Meaning, and Action
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocated_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 22. Return Codes for the Deallocate Service (continued)

Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check. The call specified one of the following:</p> <ul style="list-style-type: none"> • A Deallocate_type of deallocate_confirm, when the Sync_level for the conversation was sync_level_none • A Deallocate_type of deallocate_confirm, when the Sync_level for the conversation was syncpt • A Deallocate_type of deallocate_flush, when the Sync_level for the conversation was syncpt. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>

Table 22. Return Codes for the Deallocate Service (continued)	
Return Code	Value, Meaning, and Action
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocated_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocated_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Restrictions

Transaction programs that call the Deallocate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Error_Extract

Returns detailed information about errors indicated by APPC/MVS error return codes. Error_Extract provides reason codes and messages that describe errors that the local system finds, and error log information and a product set ID for errors that a remote TP or system finds and reports. Error_Extract returns error information only for the last APPC TP conversation service or CPI Communications call that completed processing for that conversation.

See [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,”](#) on page 77 for more information about how to use the Error_Extract service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

The figure below shows the syntax of the CALL statement for the Error_Extract service. You must code all parameters on the CALL statement in the order shown.

```
CALL ATBEES3(
    Conversation_ID,
    Service_name,
    Service_reason_code,
    Message_text_length,
    Message_text,
    Error_log_product_set_ID_length,
    Error_log_product_set_ID,
    Error_log_information_length,
    Error_log_information,
    Reason_code,
    Return_code
);
```

Figure 39. ATBEES3 - LU 6.2 Error_Extract

Parameters

The following section describes the parameters you specify when calling the Error_Extract service.

Conversation_ID

Supplied parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_ID specifies the conversation ID associated with the conversation for which you want to return problem determination information. The conversation ID, sometimes called a resource identifier, identifies a conversation to the system.

If APPC/MVS cannot establish a conversation, APPC/MVS still assigns a conversation ID to the allocate request. Your application specifies that conversation ID on this parameter.

Service_name

Returned parameter

- Type: Character
- Char Set: EBCDIC
- Length: 8 bytes

Error_Extract

Service_name specifies the name of the conversation callable service for which Error_Extract is returning error information. The name appears in the same format in which the call is coded (for example, for a Send request, the name is ATBSEND). Error_Extract returns a value on this parameter only when APPC/MVS returns an error return code to the caller of the service in error.

Service_reason_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Service_reason_code specifies the reason code for the call to the APPC/MVS conversation service in error (the service specified on the Service_Name parameter). Error_Extract returns a value on this parameter only when Error_Extract receives a zero return code.

Message_text_length

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Message_text_length indicates the total number of characters that appear in the message specified on the Message_text parameter. Error_Extract returns a value on this parameter only when APPC/MVS returns an error return code to the caller of the service in error.

Message_text

Returned parameter

- Type: Character
- Char Set: EBCDIC
- Length: 256 bytes

Message_text contains a message that describes an error on the call to the service specified on the Service_name parameter. Your application can write this message to the output stream. Error_Extract returns a value on this parameter only when APPC/MVS returns an error return code to the caller of the service in error. If APPC/MVS is the partner system that supplies this message text, the data returned for this parameter will appear as a message in the format ATB8xxxxI. See [“Error_Extract \(ATB8\) Messages” on page 343](#) for explanations of messages returned by APPC/MVS.

Error_log_product_set_ID_length

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Error_log_product_set_ID_length is the length of the value returned on the Error_Log_Product_Set_ID parameter.

- If *no* product set ID information is available from the partner system, APPC/MVS sets the value on this parameter to zero.
- If product set ID information *is* available from the partner system, APPC/MVS sets the value on this parameter to a number from 1 through 256.

If more than 256 bytes of product set ID information is available, APPC/MVS returns only the first 256 bytes of that information.

Error_log_product_set_ID

Returned parameter

- Type: Character

- Char Set: N/A
- Length: 256 bytes

Error_log_product_set_ID identifies the partner product that provided error log information, which is specified on the Error_log_information parameter for this service. APPC/MVS returns a value on this parameter only when the Return_code parameter specifies a zero value *and* the value returned on the Error_Log_Product_Set_ID_Length parameter is greater than zero. If the product set ID is more than 256 bytes long, APPC/MVS returns only the first 256 bytes of the product set ID.

For information about the format of a product set ID, see the descriptions of the Product Set ID (X'10') and the Product Identifier (X'11') MS Common Subvectors in *SNA Formats*.

Error_log_information_length

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Error_log_information_length specifies the length of the log information received from a partner TP or system. If no error log information is available from the partner TP or system, APPC/MVS sets the value on this parameter to zero.

Error_log_information

Returned parameter

- Type: Character
- Char Set: N/A
- Length: 512 bytes

Error_log_information contains a message that describes an error found by a partner system or TP. APPC/MVS returns a value in this field only when the Error_log_information_length parameter specifies a non-zero value (indicating that APPC/MVS received error log information from a partner TP or system). Error_Extract returns a value on this parameter only when APPC/MVS returns an error return code to the caller of the service in error. If APPC/MVS is the partner system that supplies this error log information, the data returned for this parameter will appear as a message in the format ASBxxxxxI or ATB7xxxxI. See “Error_Extract Error Log Information (ASB, ATB7) Messages” on page 317 for explanations of messages returned by APPC/MVS.

Reason_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Reason_code contains additional information about the result of the call to Error_Extract, when the Return_code parameter contains a value other than zero or 64 (decimal).

See “Return and Reason Codes” on page 162 for valid reason codes.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Return_code specifies the result of the call to Error_Extract. If the return_code parameter contains zero or 64 (decimal), there is no reason code. For other return codes, check the Reason_code parameter for additional information about the result of the call.

See “Return and Reason Codes” on page 162 for valid reason codes.

Return and Reason Codes

When the Error_Extract service returns control to your program, the Return_code and Reason_code parameters contain one of the following sets of values:

Return Code (decimal)	Reason Code (decimal)	Meaning and Action
0	—	<p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
4	42	<p>Meaning: A TP called the Error_Extract service to return information for an APPC/MVS service that Error_Extract does not support.</p> <p>System Action: Error_Extract does not return any error information for the specified conversation.</p> <p>Application Programmer Response: Ensure that your TP calls Error_Extract for a supported MVS TP conversation service or CPI Communications call that receives an error return code. Also ensure that the conversation ID specified on the call to Error_Extract is the same as the conversation ID specified on the call to the service that received the error return code.</p>
8	22	<p>Meaning: The conversation ID specified on the call to Error_Extract is not valid.</p> <p>System Action: Error_Extract does not return any error information for the specified conversation.</p> <p>Application Programmer Response: Validate that the conversation ID is the same as the conversation ID specified on the call to the service in error. If so, validate that the conversation was not deallocated normally. If the problem persists, ensure that the address space from which the Error_Extract was called was not cleaned up.</p>
16	8	<p>Meaning: The caller held one or more locks when it called Error_Extract.</p> <p>System Action: Error_Extract does not return any error information for the specified conversation.</p> <p>Application Programmer Response: Issue the SETLOCK assembler macro to release all held locks held before calling Error_Extract.</p>
32	16	<p>Meaning: An internal error occurred in APPC/MVS.</p> <p>System Action: Error_Extract does not return any error information for the specified conversation.</p> <p>Application Programmer Response: Ask the system programmer to contact the IBM Support Center. Tell the system programmer to provide the Support Center with the record that APPC/MVS writes to the logrec data set.</p>

Return Code (decimal)	Reason Code (decimal)	Meaning and Action
64	—	<p>Meaning: APPC/MVS is not active.</p> <p>System Action: Error_Extract cannot return any error information for the specified conversation.</p> <p>Application Programmer Response: Ask the operator to enter a START APPC command to start APPC/MVS.</p>

Restrictions

Programs that call the Error_Extract service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Flush

Equivalent to:

- LU 6.2 (MC_)Flush
- CPI Flush (CMFLUS)

Flushes the local LU's send buffer. The LU sends any information it has buffered to the partner LU.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBFLUS(
    Conversation_id,
    Notify_type,
    Return_code
);
```

Figure 40. ATBFLUS - LU 6.2 Flush

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction

- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the next section for descriptions of return codes that can be returned to a caller of Flush.

Return Codes

Valid return code values for the Return_code parameter are:

Table 23. Return Codes for the Flush Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>

Table 23. Return Codes for the Flush Service (continued)

Return Code	Value, Meaning, and Action
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: One of the following errors occurred:</p> <ul style="list-style-type: none"> • The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables. • The conversation was in send state, but the TP did not finish sending a logical record. • The conversation is not in Send or Send-pending state. • The Sync_level is set to syncpt, and the TP is in the Backout-required state. <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Flush service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Get_Attributes

Related to:

- LU 6.2 (MC_)Get_Attributes
- CPI Communications Extract_Conv_State (CMECS), Extract_Mode_Name (CMEMN), Extract_Part_LU_Name (CMEPLN), Extract_Sync_Level (CMESL)

Call the Get_Attributes service to determine certain attributes of the conversation specified by the conversation specified by the Conversation_ID parameter. This service is most useful when issued for an inbound conversation; the returned parameters provide important information about the attributes with which the conversation was allocated. Some of the data returned by Get_Attributes can be obtained only through this service. However, several fields provide information that is also available from other APPC/MVS services. For example, you can determine the conversation_type returned by Get_Attributes by calling the Get_Type service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGT6(
    Conversation_id,
    Partner_LU_name,
    Mode_name,
    Sync_level,
    Conversation_correlator,
    LUW_id,
    TP_name_length,
    TP_name,
    Local_LU_name,
    Conversation_type,
    User_id,
    Profile,
    User_token,
    Conversation_state,
    Total_timeout_value,
    Return_code
);
```

Figure 41. ATBGT6 —LU 6.2 Get_Attributes

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_ID specifies the conversation ID of the conversation for which you want to time VTAM APPCCMD requests issued during APPC/MVS conversation callable services. Specify the conversation_id that was returned from the Allocate, CMINIT, CMAACP, Get_Conversation, or Receive_Allocate call.

Partner_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes

Partner_LU_name specifies the name of the LU at which the partner program is located.

The Partner_LU_name can be one of the following:

- A VTAM generic resource name

If the partner LU is a member of a generic resource group, the Partner_LU_name might be the 1- to 8-byte generic resource name of the group.

- The network-qualified name of the partner logical unit.

The network-qualified name consists of two Type A character strings that represent the network ID and network LU name, respectively. Both strings are between 1 and 8 bytes in length, concatenated together by a period: *network_ID.network_LU_name*. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name.

- A Type A character string that is 1 to 8 bytes in length. This string represents the network LU name.

Mode_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

If Mode_name is less than 8 bytes in length, it is padded on the right with blanks.

Sync_level

Returned parameter

- Type: Integer
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not issue any protocol boundary calls and will not recognize any returned parameters relating to synchronization functions.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs may issue protocol boundary calls and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Conversation_correlator

Returned parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_correlator specifies further qualification of the LU work identifier (LUW_id) and helps restore protected resources to a consistent state following the failure of an LU, session, or conversation.

If there is no conversation correlator for the conversation, this field contains binary zeroes.

LUW_id

Returned parameter

- Type: Structure
- Char set: N/A
- Length: 26 bytes

LUW_id contains the logical unit of work (LUW) identifier. The LUW identifier is used by some logical units for accounting purposes. If the value returned on the Sync_level parameter is syncpt, a protected LUW_id is returned in this parameter. If no LUW identifier is present, this field contains binary zeroes.

TP_name_length

Returned parameter

- Type: Integer
- Length: 32 bits

TP_name_length contains the length of the data in the TP_name parameter. If the conversation_id parameter specifies an outbound conversation, this field is set to zero.

TP_name

Returned parameter

- Type: Character string
- Char Set: 00640 or Type A (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

When the conversation_id parameter specifies an inbound conversation, TP_name contains the name of the local TP for this conversation. If you called the Register_For_Allocates service to become a server of inbound allocate requests, this parameter contains the TP name specified in the FMH-5 that contained the request. When the conversation_id parameter specifies an outbound conversation, this field is not set.

Local_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Local_LU_name specifies the name of the local LU from which the conversation is initiated.

Conversation_type

Returned parameter

- Type: Integer
- Length: 32 bits

Conversation_type specifies how the data sent on this conversation is to be formatted.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

Specifies that, in this conversation, the calling program and its partner will format their data into separate logical records before sending it. Each record begins with a 2-byte length field (LL) that specifies the amount of data in the record.

1

Mapped_conversation

Specifies that, in this conversation, the calling program and its partner will rely on APPC to format the data they send.

User_id

Returned parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (left-justified if the user ID is less than 10 bytes)

If the Conversation_id parameter specifies an inbound conversation, User_id returns the user ID associated with the inbound allocate request. If the Conversation_id parameter specifies an outbound conversation, this field contains blanks.

Profile

Returned parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (left-justified if the profile name is less than 10 bytes)

If the Conversation_id parameter specifies an inbound conversation, profile contains the RACF group name associated with the inbound allocate request. When the Conversation_id parameter specifies an outbound conversation, this field contains blanks.

User_token

Returned parameter

- Type: Structure
- Char Set: N/A
- Length: 80 bytes

If the Conversation_id parameter specifies an inbound conversation, User_token contains the RACF UTOKEN that identifies the user associated with the inbound allocate. This token is encrypted.

If the conversation_id parameter specifies an outbound conversation, this field is blanks. No UTOKEN is returned for an outbound conversation.

Conversation_state

Returned parameter

- Type: Integer
- Length: 32 bits

Conversation_state specifies the current state of the conversation, which is one of the following:

Value

Conversation State

2	Initialize
3	Send
4	Receive
5	Send-Pending
6	Confirm
7	Confirm-Send
8	Confirm-Deallocate
9	Defer-Receive
10	Defer-Deallocate
11	Sync-Point
12	Sync-Point-Send
13	Sync-Point-Deallocate

For descriptions, see [“APPC/MVS TP Conversation States” on page 45.](#)

Total_Timeout_Value

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Specifies IN SECONDS the APPC conversational timeout value that was set for this conversation. This timeout value may have been set by either the Allocate or Set_Timeout Service earlier. If set, this parameter combines the Timeout_Value_Minutes and Timeout_Value_Seconds values together. The minimum value to be returned is zero (the timer has not been set or has been reset off). The maximum value that can be returned is 86459 seconds (Timeout_Value_Minutes maximum value + Timeout_Value_Seconds maximum value, converted into seconds).

Return_code

Returned parameter

- Type: Integer
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See [“Return Codes” on page 170](#) for descriptions of return codes that can be returned to a caller of Get_Attributes.

Return Codes

Valid return code values for the Return_code parameter are:

Table 24. Return Codes for the Get_Attributes Service

Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

TPs that call the Get_Attributes service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Get_Conversation

Equivalent to:

- (No LU 6.2 equivalent)
- CPI Accept_Conversation (CMACCP)

Get_Conversation

To be used by an allocated TP to return the conversation ID that the TP will use to reference the conversation on which it was attached, and also to return information pertaining to the specified conversation.

Get_Conversation cannot be issued once activity has begun on the conversation. It is only allowable as the very first call issued for the conversation. After a call has been issued on the conversation, subsequent calls to the Get_Conversation service will return a return code of Program_State_Check.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGETC(  
    Conversation_id,  
    Conversation_type,  
    Partner_LU_name,  
    Mode_name,  
    Sync_level,  
    Conversation_correlator,  
    Return_code  
);
```

Figure 42. ATBGETC - Get_Conversation

Parameters

Conversation_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Conversation_type

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the call is issued.

Valid values for this parameter are:

Value	Meaning
-------	---------

0

Basic_conversation

1

Mapped_conversation

Partner_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes

Partner_LU_name specifies the name of the LU at which the partner program is located.

The value returned for Partner_LU_name is the network-qualified name of the partner logical unit. The network-qualified name consists of two Type A character strings that represent the network ID and network LU name, respectively. Both strings are between 1 and 8 bytes in length, concatenated together by a period: *network_ID.network_LU_name*.

If the partner LU is a member of a generic resource group, the network-LU-name portion of Partner_LU_name might be the 1- to 8-byte generic resource name of the group.

Mode_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

If Mode_name is less than 8 bytes in length, it is padded on the right with blanks.

Sync_level

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value**Meaning****0**

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not issue any protocol boundary calls and will not recognize any returned parameters relating to synchronization functions.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs may issue protocol boundary calls and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Conversation_correlator

Returned parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_correlator specifies further qualification of the LU work identifier (LUW_id) and helps restore protected resources to a consistent state following the failure of an LU, session, or conversation.

If there is no conversation correlator for the conversation, this field contains all binary zeroes.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See “Return Codes” on [page 174](#) for descriptions of return codes that can be returned to a caller of Get_Conversation.

Return Codes

Valid return code values for the Return_code parameter are:

<i>Table 25. Return Codes for the Get_Conversation Service</i>	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 25. Return Codes for the Get_Conversation Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Get_Conversation service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Get_TP_Properties

Note: You cannot use the Error_Extract conversation service to diagnose errors in calls to the Get_TP_Properties service.

Equivalent to:

- LU 6.2 Get_TP_Properties
- (No CPI equivalent)

Returns information pertaining to the transaction program issuing the call.

This service requires TP resources to be associated with the calling address space to complete successfully. This service should be issued after issuing one of the following services that will associate TP resources to the calling address space:

- CMINIT
- Allocate
- Register_For_Allocate
- Define_Local_TP

This service may also be issued:

- After the transaction scheduler XCF message user routine has received an Allocate TP request message to obtain information pertaining to the transaction program represented by the received message.
- When an application program begins executing as the result of a transaction scheduler receiving and processing an Allocate TP request.
- Before the Get_Conversation or CPI Accept_Conversation (CMACCP) services.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode

Cross memory mode:	PASN = HASN = SASN or PASN \neg = HASN \neg = SASN
Amode:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGTP4(
    Own_TP_name_length,
    Own_TP_name,
    Own_fully_qualified_LU_name,
    User_id,
    Profile,
    LUW_id,
    Vote_Read_Only_Permitted,
    Wait_For_Outcome,
    Action_If_Problems,
    Return_code
);
```

Figure 43. ATBGTP4 - LU 6.2 Get_TP_Properties

Parameters

Own_TP_name_length

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Own_TP_name_length specifies the length of data contained in the Own_TP_name parameter. If the Own_TP_name parameter does not contain a TP_name on return from this service, Own_TP_name_length is set to zero.

Own_TP_name

Returned parameter

- Type: Character string
- Char Set: 00640 or Type A
- Length: 64 bytes

Own_TP_name specifies the name of the local program as specified in the FMH-5 allocation request. This parameter will only contain a return value if the local program was started as the result of an attach request from a partner program. If this is not the case, there is no TP name to be returned, and Own_TP_Name_Length is set to zero.

Own_fully_qualified_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes

Own_fully_qualified_LU_name specifies the network-qualified name of the local logical unit.

User_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 10 bytes

User_id specifies the userid that is associated with the caller's address space. If the address space contains a scheduled transaction program, the User_id parameter contains the userid that accompanied the inbound transaction program request.

Profile

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 10 bytes

Profile specifies the RACF group name associated with the caller's address space. If the address space contains a scheduled transaction program, the profile parameter contains the RACF group name that accompanied the inbound transaction program request. If the inbound request did not include a profile, the profile parameter contains the default profile for the transaction program that issued the request.

LUW_id

Returned parameter

- Type: Structure
- Char Set: N/A
- Length: 26 bytes

LUW_id specifies the logical unit of work (LUW) identifier. The LUW identifier is used by some logical units for accounting purposes. If no LUW identifier is present, this field will contain binary zeroes.

Note: A LUW_id for protected resources (such as a protected conversation) is called a protected LUW_id. Because a TP might have more than one protected LUW_id associated with it, APPC/MVS does not return a protected LUW_id in this parameter, even if one exists and no LUW_id for unprotected resources exists. To obtain the protected LUW_id of a protected conversation, use the Get_Attributes service.

Vote_Read_Only_Permitted

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Vote_Read_Only_Permitted specifies whether the local LU may vote read only in a sync point operation. This syncpt option can be set by calling the Set_Syncpt_Options service.

Possible returned values for this parameter are:

Value**Meaning**

1

NO

Specifies that voting read only in a sync point operation is not allowed.

2

YES

Specifies that voting read only in a sync point operation is allowed.

Wait_For_Outcome

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Wait_For_Outcome specifies whether the outcome of the sync point operation at all subordinate resources in the distributed transaction must be known before control is returned to the program. This syncpt option can be set by calling the Set_Syncpt_Options service.

Possible returned values for this parameter are:

Value	Meaning
1	NO
2	YES

Action_If_Problems

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Action_If_Problems specifies the action to be taken with protected resources if the LU learns of a problem at a point in the sync point operation when it does not know whether to commit or backout. This syncpt option can be set by calling the Set_Syncpt_Options service.

Possible returned values for this parameter are:

Value	Meaning
1	Commit
2	Backout

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See “Return Codes” on page 179 for descriptions of return codes that can be returned to a caller of Get_TP_Properties.

Return Codes

Valid return code values for the Return_code parameter are:

Table 26. Return Codes for the Get_TP_Properties Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The program called a service under conditions in which the call is not valid; for example:</p> <ul style="list-style-type: none"> • APPC/MVS might not recognize the program as a local TP. • APPC/MVS might have encountered temporary environmental conditions that prevent it from obtaining the requested information. <p>The program should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged.</p> <p>Application Programmer Response: Design the program to re-issue the call; this error condition might be temporary.</p>

Restrictions

Transaction programs that call the Get_TP_Properties service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Get_Type

Equivalent to:

- LU 6.2 Get_Type
- CPI Extract_Conv_Type (CMECT)

Returns the type of conversation to which the specified conversation ID is assigned.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGETT(  
    Conversation_id,  
    Conversation_type,  
    Return_code  
);
```

Figure 44. ATBGETT - LU 6.2 Get_Type

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Conversation_type

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the call is issued.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

1

Mapped_conversation

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A

- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See “Return Codes” on page 181 for descriptions of return codes that can be returned to a caller of Get_Type.

Return Codes

Valid return code values for the Return_code parameter are:

Table 27. Return Codes for the Get_Type Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Get_Type service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Post_on_Receipt

Requests to be notified when data or status is ready to be received for a specified conversation. Specifically, Post_on_Receipt notifies the caller (through an ECB the caller specifies) for the following situations:

- A complete logical record is available to be received from a partner program
- Conversation status (control information) is available

Post_on_Receipt

- A non-zero return code is available to be received because of an action taken by the partner program (such as deallocating the conversation).

When Post_on_Receipt posts the specified ECB, the caller can determine which of the preceding information is available by calling the Receive_and_Wait or Receive_Immediate service and checking the returned parameters.

You can call the Post_on_Receipt service in Receive state only, and it does not alter the conversation state. This function only applies to basic conversations.

Asynchronous Processing

To be notified asynchronously when data or status is ready for you to receive on a conversation, you can use the Post_on_Receipt service. APPC/MVS posts the ECB specified by the ECB_address parameter when data or status, or both, is available to be received. The caller is free to call other conversation services (such as Request_to_Send) while APPC/MVS processes the Post_on_Receipt request asynchronously.

Your call to Post_on_Receipt remains in effect until the specified ECB is posted, or the call is cancelled. Thereafter, to obtain subsequent notification of data or status to be received, issue a new call to the Post_on_Receipt service.

APPC/MVS cancels an active Post_on_Receipt request if the caller issues any of the following services on the same conversation:

- Receive_Immediate
- Receive_and_Wait
- Deallocate (with the deallocate_type parameter set to deallocate_abend)
- Send_Error.

You can call the Post_on_Receipt service any number of times on a given conversation. However, if you call the Post_on_Receipt service multiple times before the specified ECB is posted, only one post can occur. In this situation, APPC/MVS posts the ECB that was specified on the most recent call to the Post_on_Receipt service.

Receiving Asynchronous Notification

When asynchronous processing is complete, the POST completion code in the ECB is the return code for the service. Post_on_Receipt posts the ECB with a completion code of 0 when information is available. You can then determine what this information is by calling either the Receive_and_Wait or Receive_Immediate services. These services return the following information about the specified conversation:

- There is a complete logical record is available to be received from the partner program (see the Data_received parameter returned by Receive_and_Wait or Receive_Immediate).
- There is a change in conversation status, to one of the following (as shown in the Status_received parameter returned by Receive_and_Wait or Receive_Immediate):
 - Send_received
 - Confirm_received
 - Confirm_send_received
 - Confirm_deallocate_received.

Note that Post_on_Receipt does not notify the caller for a Request_To_Send Received condition.

- A previous call by the partner program on this conversation caused one of the following non-zero return codes (indicated in the Return_code parameter of Receive_and_Wait or Receive_Immediate):
 - 3 - Conversation_type_mismatch
 - 5 - Pip_not_specified_correctly
 - 6 - Security_not_valid

8 - Sync_lvl_not_supported_pgm
 9 - TPN_not_recognized
 10 - TP_not_available_no_retry
 11 - TP_not_available_retry
 17 - Deallocated_abend
 18 - Deallocated_normal
 21 - Program_error_no_trunc
 22 - Program_error_purging
 23 - Program_error_trunc
 26 - Resource_failure_no_retry
 27 - Resource_failure_retry
 30 - Deallocated_abend_SVC
 31 - Deallocated_abend_timer
 32 - SVC_error_no_trunc
 33 - SVC_error_purging
 34 - SVC_error_trunc

Post_on_Receipt posts the caller's ECB with a non-zero POST return code if the service ends unsuccessfully. For example, if the APPC address space is cancelled before information for the conversation becomes available, Post_on_Receipt posts the caller's ECB with a return code of product_specific_error.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller in the primary address space.

Format

```
CALL ATBPOR2(
    Conversation_id,
    ECB_address,
    Return_code
);
```

Figure 45. ATBPOR2 - Post_on_Receipt

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

ECB_address

Supplied parameter

- Type: Address
- Char Set: N/A
- Length: 32 bits

ECB_address specifies the address of a fullword that specifies the address of the ECB to be posted. The ECB must reside in the caller's home address space.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the result of the call. See the next section for descriptions of return codes that can be returned to a caller of Post_On_Receipt.

Return Codes

Valid return code values for the Return_code parameter are:

<i>Table 28. Return Codes for the Post_on_Receipt Service</i>	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 28. Return Codes for the Post_on_Receipt Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

When using this service, observe the following restrictions:

- Call Post_on_Receipt only for basic conversations; do not call the service for mapped conversations. If you call the service for a mapped conversation, the service returns a return code of program_parameter_check.
- If running in task mode, do not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).
- Call Post_on_Receipt in receive state only. If you call the service in a state other than Receive state, the service returns a return code of program_state_check.
- Do not call the Post_on_Receipt service when another callable service is outstanding on the specified conversation. If you call the service while another service is in effect, Post_on_Receipt returns a return code of product_specific_error.

Prepare_to_Receive

Equivalent to:

- LU 6.2 (MC_)Prepare_to_Receive
- CPI Prep_To_Receive (CMPTR)

Changes the conversation from Send to Receive state in preparation to receive data.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBPTR(  
    Conversation_id,  
    Prepare_to_receive_type,  
    Locks,  
    Notify_type,  
    Return_code  
);
```

Figure 46. ATBPTR - LU 6.2 Prepare_to_Receive

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Prepare_to_receive_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Prepare_to_receive_type specifies the type of prepare_to_receive function to be performed on this call.

Valid values for this parameter are:

Value

Meaning

0

Prep_to_receive_sync_level

Specifies to perform the Prepare_to_receive based on the synchronization level in effect for the conversation:

- If the synchronization level is None, execute the function of the Flush call and enter Receive state.
- If the synchronization level is Confirm, execute the function of the Confirm call and if it is successful, enter Receive state.
- If the synchronization level is Syncpt, enter Defer_receive state until the program issues either a Commit call, or a Confirm or Flush call, for this conversation. Upon successful completion of the Commit, Confirm, or Flush call, the TP enters Receive state.

1

Prep_to_receive_flush

Specifies to execute the function of the Flush call and enter Receive state.

2

Prep_to_receive_confirm

Specifies to execute the function of the Confirm call and if it is successful, enter Receive state.

Locks

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Locks specifies when control is to be returned to the program following execution of the confirmation function of this call.

This parameter is significant only when the Prepare_to_receive_type parameter contains a value of Prep_to_receive_confirm, or a value of Prep_to_receive_sync_level and the synchronization level of the conversation is confirm or syncpt. Otherwise, this parameter has no meaning and is ignored.

Valid values for this parameter are:

Value

Meaning

100

Short

Specifies to return control when an affirmative reply is received, as follows:

- When the synchronization level is confirm, return control from execution of this call when a Confirmed reply is received.

101

Long

Specifies to return control when information, such as data, is received from the partner program following an affirmative reply, as follows:

- When the synchronization level is confirm, return control from execution of this call when information is received following a Confirmed reply.
- When the synchronization level is syncpt, immediately return control from execution of this call; return control from execution of a subsequent Confirm or Commit call when information is received following a corresponding Confirmed or Syncpt reply.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during

Prepare_to_Receive

asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Prepare_To_Receive.

Return Codes

If the value of the Prepare_to_receive_type parameter contains the value Prep_to_receive_flush, or it contains the value Prep_to_receive_sync_level and the synchronization level of the conversation is None, possible values for Return_code are:

Decimal Value

Meaning

0

OK

20

Product_specific_error

24

Program_parameter_check

25

Program_state_check

If the value of the Prepare_to_receive_type parameter contains the value Prep_to_receive_confirm, or it contains the value Prep_to_receive_sync_level and the synchronization level of the conversation is Confirm, possible values of Return_code are:

Decimal Value

Meaning

0

OK

3

Conversation_type_mismatch

5

PIP_not_specified_correctly

6

Security_not_valid

8

Sync_lvl_not_supported_pgm

9

TPN_not_recognized

10

TP_not_available_no_retry

11

TP_not_available_retry

17	Deallocate_abend
20	Product_specific_error
22	Program_error_purging
24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_SVC
31	Deallocated_abend_timer
33	SVC_error_purging

If the value of the Prepare_to_receive_type parameter contained the value Prep_to_receive_confirm and the synchronization level of the conversation is syncpt, possible values of Return_code are:

Decimal Value

Meaning

0	OK
3	Conversation_type_mismatch
5	PIP_not_specified_correctly
6	Security_not_valid
8	Sync_lvl_not_supported_pgm
9	TPN_not_recognized
10	TP_not_available_no_retry
11	TP_not_available_retry
17	Deallocate_abend
20	Product_specific_error
22	Program_error_purging
24	Program_parameter_check

Prepare_to_Receive

25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_SVC
31	Deallocated_abend_timer
33	SVC_error_purging
100	Take_backout
130	Deallocated_abend_bo
131	Deallocated_abend_svc_bo (basic conversations only)
132	Deallocated_abend_timer_bo (basic conversations only)
133	Resource_failure_no_retry_bo
134	Resource_failure_retry_bo

If the value of the Prepare_to_receive_type parameter contained the value Prep_to_receive_sync_level and the synchronization level of the conversation is syncpt, possible values of Return_code are:

Decimal Value Meaning

0	OK
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check

The following table describes *all* of the possible return codes for Prepare_To_Receive.

Table 29. Return Codes for the Prepare_to_Receive Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)

Return Code	Value, Meaning, and Action
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)	
Return Code	Value, Meaning, and Action
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocate_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)

Return Code	Value, Meaning, and Action
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)	
Return Code	Value, Meaning, and Action
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocated_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocated_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)

Return Code	Value, Meaning, and Action
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
132	<p>Value: Deallocated_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 29. Return Codes for the Prepare_to_Receive Service (continued)

Return Code	Value, Meaning, and Action
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

Transaction programs that call the Prepare_to_Receive service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Receive_Immediate

Equivalent to:

- LU 6.2 (MC_)Receive_Immediate
- CPI Receive (CMRCV)

Receives any information that is available on the specified conversation but does not wait for information to arrive. Control is returned to the program with an indication of whether any information was received and, if so, the type of information.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit

ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space, except for the <i>buffer</i> parameter.

Format

```
CALL ATBRCVI(  
    Conversation_id,  
    Fill,  
    Receive_length,  
    Access_token,  
    Buffer,  
    Status_received,  
    Data_received,  
    Request_to_send_received,  
    Return_code  
);
```

Figure 47. ATBRCVI - LU 6.2 Receive_Immediate

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Fill

Supplied

- Type: Integer
- Char Set: N/A
- Length: 32 bits

In a basic conversation, specifies whether the program is to receive data in terms of the logical record format of the data.

Valid values for this parameter are:

Value

Meaning

0

LL

Specifies the program is to receive one logical record, or whatever portion of the logical record that is available, up to the length specified.

1

Buffer

Specifies the program is to receive data independent of its logical record format, up to the length specified.

This parameter has no effect on a mapped conversation, but must contain a valid value.

Receive_Immediate

Receive_length

Supplied/Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Receive_length specifies the maximum amount of data that the program is to receive. When control is returned to the program, this parameter contains the actual amount of data that the program received up to the maximum. If the program receives information other than data (that is, a control signal), this parameter remains unchanged.

No value is returned in Receive_length if Data_received is not returned to the program or if Data_received indicates No_data_received.

Access_token

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Access_token specifies the access list entry token (ALET) of the address space or data space in which the buffer resides for Receive_immediate calls.

APPC/MVS always uses access_token together with the address of the buffer to resolve addressing to the transaction program's data. To specify that the buffer address passed should not be ALET qualified, an Access_token value of zero should be supplied. APPC/MVS will then consider the buffer to reside in the primary address space of the caller.

The Access_token can:

- Represent an entry on the dispatchable unit access list (DU-AL)
- Represent an entry on the caller's primary address access list (PASN-AL), *only if the entry points to a SCOPE=COMMON data space.*

The Access_token cannot:

- Be the value 1 (which indicates "secondary ASID")
- Represent an entry on the caller's PASN-AL that *does not* point to a SCOPE=COMMON data space.

For more information about ALETs for SCOPE=COMMON data spaces, see [“Features of the MVS-Specific Services”](#) on page 32.

Buffer

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 0-2,147,483,647 bytes

Buffer specifies the buffer that is to contain the data to be received. The call supplies the buffer that is to contain the data. APPC/MVS returns the data in the supplied buffer. This data can consist entirely of data (for mapped conversations) or logical records (for basic conversations).

If the data consists of logical records, each such record consists of a two-byte length field followed by a data field; the length of the data field can range from zero to 32,765 bytes. The length of the record includes the two-byte length field; therefore, logical-record length values of X'0000', X'0001', X'8000', and X'8001' are not valid.

No value is returned in Buffer if Data_received is not returned to the program or if Data_received indicates No_data_received.

Status_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Status_received specifies what control information was received.

Valid return values for this parameter are:

Value**Meaning****0**

No_status_received

1

Send_received

Indicates that the partner program has entered Receive state, placing the local program in Send state. The local program may now issue a Send_data call.

2

Confirm_received

Indicates that the partner program has issued a confirmation request, requesting the local program to respond with a Confirmed call. The program may respond instead by issuing a call other than Confirmed, such as Send_error.

3

Confirm_send_received

Indicates that the partner program executed the prepare to receive function with one of the following:

- A type of confirm
- A type of sync_level and the synchronization level is confirm
- A type of sync_level and the synchronization level is syncpt, followed by a confirmation request.

The local program may respond by issuing a Confirmed call, or by issuing another call such as Send_error.

4

Confirm_dealloc_received

Indicates the partner program executed the deallocate function with a type of confirm; or with a type of sync_level and the synchronization level is confirm. The local program may respond by issuing a Confirmed call, or by issuing another call such as Send_error.

For a conversation with synchronization level set to syncpt, the following values are also valid:

Value**Meaning****5**

Take_syncpt

Indicates that the remote program has issued a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. When appropriate, the local program may respond by issuing a call other than Commit, such as Backout or Send_Error, which causes the transaction to back out.

6

Take_syncpt_send

Receive_Immediate

Indicates that the remote program executed the Prepare_To_Receive function with a Prepare_To_Receive_Type of sync_level and the synchronization level set to syncpt followed by a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. The local program should respond with a Commit call to commit all protected resources throughout this transaction. When appropriate, the local program may respond by issuing a call other than Commit, such as Backout or Send_Error.

7

Take_syncpt_dealloc

Indicates that the remote program executed the Deallocate function with a deallocate_type of sync_level and the synchronization level set to syncpt followed by a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. The local program should respond with a Commit call to commit all protected resources throughout this transaction and have the conversation deallocated. When appropriate, the local program may respond by issuing a call other than Commit such as Backout or Send_Error.

Data_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Data_received specifies what type of data was received.

Valid return values for this parameter are:

Value

Meaning

0

No_data_received

No data was received.

1

Data_received

When the conversation is basic and the value of the Fill parameter was set to Buffer, this return value indicates that data (independent of its logical record format) was received.

2

Complete_data_received

- For a basic conversation

When the value of the Fill parameter was set to LL, this return value indicates that a complete logical record, or the last remaining portion of it, was received.

- For a mapped conversation

This return value indicates that a complete data record, or the last remaining portion of it, was received.

3

Incomplete_data_received

- For a basic conversation

When the value of the Fill parameter was set to LL, this return value indicates that less than a complete logical record was received. The local program must issue one or more additional receive calls to receive the remainder of the data.

- For a mapped conversation

This return value indicates that less than a complete data record was received. The local program must issue one or more additional receive calls to receive the remainder of the data.

If Return_code indicates any value other than OK or Deallocated_Normal, a value is not returned in Data_received.

Request_to_send_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Request_to_send_received specifies whether or not Request_to_send notification has been received.

Valid return values for this parameter are:

Value

Meaning

0

Request_to_send_not_received

Indicates that Request_to_send notification has not been received

1

Request_to_send_received

The partner program has issued a Request_to_send, requesting the local program to enter Receive state.

If Return_code indicates Program_parameter_check or Program_state_check, a value is not returned in Request_to_send_received.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Receive_Immediate.

Return Codes

The following table lists all of the possible return codes for Receive_Immediate:

Table 30. Return Codes for the Receive_Immediate Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)

Return Code	Value, Meaning, and Action
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Ask the partner system programmer to provide a valid partner TP name. When requesting the allocate, specify the valid partner TP name.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)

Return Code	Value, Meaning, and Action
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocate_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
18	<p>Value: Deallocate_normal</p> <p>Meaning: A partner TP called the Deallocate service for a basic or mapped conversation. The request specified a Deallocate_type of Deallocate_sync_level or Deallocate_flush.</p> <p>System Action: The system returns this return code to the local TP when it calls a service while the conversation is in Receive state.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)	
Return Code	Value, Meaning, and Action
21	<p>Value: Program_error_no_trunc</p> <p>Meaning: Indicates one of the following:</p> <ul style="list-style-type: none"> • A partner TP called the Send_Error service for a mapped conversation. The conversation for the local TP was in Send state. No truncation occurs at the mapped conversation protocol boundary. • A partner TP called Send_Error for a basic conversation. The conversation was in Send state. The call did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a TP calls Send_Error either before sending any logical records or after sending a complete logical record. <p>System Action: The system returns this return code to the local TP when it calls the Receive service, before the TP receives any data records or after it receives one or more data records.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
23	<p>Value: Program_error_trunc</p> <p>Meaning: The partner TP called the Send_Error service for a basic conversation. The conversation for the partner TP was in Send state, and the call truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a TP begins sending a logical record and then makes a Send_error call before sending the complete logical record.</p> <p>System Action: The system returns this return code to the local TP on a Receive call that occurs after the TP receives the truncated logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
28	<p>Value: Unsuccessful</p> <p>Meaning: No data was available to receive.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)

Return Code	Value, Meaning, and Action
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocated_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
32	<p>Value: SVC_error_no_trunc</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation was in Send state, and the call did not truncate a logical record.</p> <p>System Action: The system returns this return code to the caller of the Receive service. The system does not return this return code to callers of the CPI Communications Send_Error call.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
34	<p>Value: SVC_error_trunc</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in send state, and the call truncated a logical record. Truncation occurs when a program begins sending a logical record and calls the Send_Error service before the complete record is sent.</p> <p>System Action: The system returns this return code when the local TP calls the Receive service to receive the truncated logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)

Return Code	Value, Meaning, and Action
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 30. Return Codes for the Receive_Immediate Service (continued)	
Return Code	Value, Meaning, and Action
132	<p>Value: Deallocated_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

Transaction programs that call the Receive_Immediate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Receive_and_Wait

Equivalent to:

- LU 6.2 (MC_)Receive_and_Wait

- CPI Receive (CMRCV)

Waits for information to arrive on the conversation and then receives the information. If information is already available, the program receives it without waiting.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space, except for the <i>buffer</i> parameter.

Format

```
CALL ATBRCVW(  
    Conversation_id,  
    Fill,  
    Receive_length,  
    Access_token,  
    Buffer,  
    Status_received,  
    Data_received,  
    Request_to_send_received,  
    Notify_type,  
    Return_code  
);
```

Figure 48. ATBRCVW - LU 6.2 Receive and Wait

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Fill

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

In a basic conversation, specifies whether the program is to receive data in terms of the logical record format of the data.

Valid values for this parameter are:

Value	Meaning
-------	---------

Receive_and_Wait

0

LL

specifies the program is to receive one logical record, or whatever portion of the logical record that is available, up to the length specified.

1

Buffer

specifies the program is to receive data independent of its logical record format, up to the length specified.

This parameter has no effect on a mapped conversation, but must contain a valid value.

Receive_length

Supplied/Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Receive_length specifies the maximum amount of data that the program is to receive. When control is returned to the program, this parameter contains the actual amount of data that the program received up to the maximum. If the program receives information other than data (that is, a control signal), this parameter remains unchanged.

No value is returned in Receive_length if Data_received is not returned to the program or if Data_received indicates No_data_received.

Access_token

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Access_token specifies the Access List Entry Token (ALET) of the address space or data space in which the buffer resides for Receive_and_wait.

APPC/MVS always uses access_token in conjunction with the address of the buffer in order to resolve addressing to the transaction program's data. To specify that the buffer address passed should not be ALET qualified, an Access_token value of zero should be supplied. APPC/MVS will then consider the buffer to reside in the primary address space of the caller.

The Access_token can:

- Represent an entry on the dispatchable unit access list (DU-AL)
- Represent an entry on the caller's primary address access list (PASN-AL), *only if the entry points to a SCOPE=COMMON data space.*

The Access_token cannot:

- Be the value 1 (which indicates "secondary ASID")
- Represent an entry on the caller's PASN-AL that *does not* point to a SCOPE=COMMON data space.

For more information about ALETs for SCOPE=COMMON data spaces, see [“Features of the MVS-Specific Services”](#) on page 32.

Buffer

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 0-2,147,483,647 bytes

Buffer specifies the buffer that is to contain the data to be received. The call supplies the buffer that is to contain the data. APPC/MVS returns the data in the supplied buffer. This data can consist entirely of data (for mapped conversations) or logical records (for basic conversations).

If the data consists of logical records, each such record consists of a two-byte length field followed by a data field; the length of the data field can range from zero to 32,765 bytes. The length of the record includes the two-byte length field; therefore, logical-record length values of X'0000', X'0001', X'8000', and X'8001' are not valid.

No value is returned in Buffer if Data_received is not returned to the program or if Data_received indicates No_data_received.

Status_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Status_received specifies what control information was received.

Valid return values for this parameter are:

Value

Meaning

0

No_status_received

1

Send_received

Indicates that the partner program has entered Receive state, placing the local program in Send state. The local program may now issue a Send_data call.

2

Confirm_received

Indicates that the partner program has issued a confirmation request, requesting the local program to respond with a Confirmed call. The program may respond instead by issuing a call other than Confirmed, such as Send_error.

3

Confirm_send_received

Indicates that the partner program executed the prepare to receive function with one of the following:

- A type of confirm
- A type of sync_level and the synchronization level is confirm
- A type of sync_level and the synchronization level is syncpt, followed by a confirmation request.

The local program may respond by issuing a Confirmed call, or by issuing another call such as Send_error.

4

Confirm_dealloc_received

Indicates the partner program executed the deallocate function with a type of confirm; or with a type of sync_level and the synchronization level is confirm. The local program may respond by issuing a Confirmed call, or by issuing another call such as Send_error.

For a conversation with synchronization level set to syncpt, the following values are also valid:

Value

Meaning

5

Take_syncpt

Indicates that the remote program has issued a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. When appropriate, the local program may respond by issuing a call other than Commit, such as Backout or Send_Error, which causes the transaction to back out.

6

Take_syncpt_send

Indicates that the remote program executed the Prepare_To_Receive function with a Prepare_To_Receive_Type of sync_level and the synchronization level set to syncpt followed by a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. The local program should respond with a Commit call to commit all protected resources throughout this transaction. When appropriate, the local program may respond by issuing a call other than Commit, such as Backout or Send_Error.

7

Take_syncpt_dealloc

Indicates that the remote program executed the Deallocate function with a deallocate_type of sync_level and the synchronization level set to syncpt followed by a syncpoint request, requesting the local program to respond with a Commit call to commit all protected resources throughout this transaction. The local program should respond with a Commit call to commit all protected resources throughout this transaction and have the conversation deallocated. When appropriate, the local program may respond by issuing a call other than Commit such as Backout or Send_Error.

Data_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Data_received specifies what type of data was received.

Valid return values for this parameter are:

Value

Meaning

0

No_data_received

No data was received

1

Data_received

When the conversation is basic and the value of the Fill parameter was set to Buffer, this return value indicates that data (independent of its logical record format) was received.

2

Complete_data_received

- For a basic conversation

When the value of the Fill parameter was set to LL, this return value indicates that a complete logical record, or the last remaining portion thereof, was received.

- For a mapped conversation

This return value indicates that a complete data record, or the last remaining portion thereof, was received.

3

Incomplete_data_received

- For a basic conversation

When the value of the Fill parameter was set to LL, this return value indicates that less than a complete logical record was received. The local program must issue one or more additional receive calls to receive the remainder of the data.

- For a mapped conversation

This return value indicates that less than a complete data record was received. The local program must issue one or more additional receive calls to receive the remainder of the data.

If Return_code indicates any value other than OK or Deallocated_Normal, a value is not returned in Data_received.

Request_to_send_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Request_to_send_received specifies whether or not Request_to_send notification has been received.

Valid return values for this parameter are:

Value**Meaning****0**

Request_to_send_not_received

Indicates that Request_to_send notification has not been received

1

Request_to_send_received

Indicates that the partner program has issued a Request_to_send, requesting the local program to enter Receive state.

If Return_code indicates Program_parameter_check or Program_state_check, a value is not returned in Request_to_send_received.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword

Receive_and_Wait

binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Receive_and_Wait.

Return Codes

If Receive_and_Wait is called in Send state, possible values of Return_code are:

Decimal Value

Meaning

0

OK

3

Conversation_type_mismatch

5

PIP_not_specified_correctly

6

Security_not_valid

8

Sync_lvl_not_supported_pgm

9

TPN_not_recognized

10

TP_not_available_no_retry

11

TP_not_available_retry

17

Deallocated_abend

18

Deallocated_normal

20

Product_specific_error

21

Program_error_no_trunc

22

Program_error_purging

24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_svc
31	Deallocated_abend_timer
32	SVC_error_no_trunc
33	SVC_error_purging
100	Take_backout
130	Deallocated_abend_bo
131	Deallocated_abend_svc_bo (basic conversations only)
132	Deallocated_abend_timer_bo (basic conversations only)
133	Resource_failure_no_retry_bo
134	Resource_failure_retry_bo

Note: Return codes 100 through 134 are possible values only for conversations with a synchronization level of syncpt.

If Receive_and_Wait is called in Send-pending state, possible values of Return_code are:

Decimal Value
Meaning

0	OK
17	Deallocated_abend
18	Deallocated_normal
20	Product_specific_error
21	Program_error_no_trunc
22	Program_error_purging
24	Program_parameter_check
25	Program_state_check

Receive_and_Wait

26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_svc
31	Deallocated_abend_timer
32	SVC_error_no_trunc
33	SVC_error_purging
100	Take_backout
130	Deallocated_abend_bo
131	Deallocated_abend_svc_bo (basic conversations only)
132	Deallocated_abend_timer_bo (basic conversations only)
133	Resource_failure_no_retry_bo
134	Resource_failure_retry_bo

Note: Return codes 100 through 134 are possible values only for conversations with a synchronization level of syncpt.

If Receive_and_Wait is called in Receive state, possible values of Return_code are:

Decimal Value	Meaning
----------------------	----------------

0	OK
3	Conversation_type_mismatch
5	PIP_not_specified_correctly
6	Security_not_valid
8	Sync_lvl_not_supported_pgm
9	TPN_not_recognized
10	TP_not_available_no_retry
11	TP_not_available_retry
17	Deallocated_abend
18	Deallocated_normal

20	Product_specific_error
21	Program_error_no_trunc
22	Program_error_purging
23	Program_error_trunc
24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_SVC
31	Deallocated_abend_timer
32	SVC_error_no_trunc
33	SVC_error_purging
34	SVC_error_trunc
100	Take_backout
130	Deallocated_abend_bo
131	Deallocated_abend_svc_bo (basic conversations only)
132	Deallocated_abend_timer_bo (basic conversations only)
133	Resource_failure_no_retry_bo
134	Resource_failure_retry_bo

Note: Return codes 100 through 134 are possible values only for conversations with a synchronization level of syncpt.

The following table lists all of the possible return codes for Receive_and_Wait:

Table 31. Return Codes for the Receive_and_Wait Service

Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)

Return Code	Value, Meaning, and Action
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Ask the partner system programmer to provide a valid partner TP name. When requesting the allocate, specify the valid partner TP name.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocate_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
18	<p>Value: Deallocate_normal</p> <p>Meaning: A partner TP called the Deallocate service for a basic or mapped conversation. The request specified a Deallocate_type of Deallocate_sync_level or Deallocate_flush.</p> <p>System Action: The system returns this return code to the local TP when it calls a service while the conversation is in Receive state.</p> <p>Application Programmer Response: None required.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)

Return Code	Value, Meaning, and Action
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
21	<p>Value: Program_error_no_trunc</p> <p>Meaning: Indicates one of the following:</p> <ul style="list-style-type: none"> • A partner TP called the Send_Error service for a mapped conversation. The conversation for the local TP was in Send state. No truncation occurs at the mapped conversation protocol boundary. • A partner TP called Send_Error for a basic conversation. The conversation was in Send state. The call did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a TP calls Send_Error either before sending any logical records or after sending a complete logical record. <p>System Action: The system returns this return code to the local TP when it calls the Receive service, before the TP receives any data records or after it receives one or more data records.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
23	<p>Value: Program_error_trunc</p> <p>Meaning: The partner TP called the Send_Error service for a basic conversation. The conversation for the partner TP was in Send state, and the call truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a TP begins sending a logical record and then makes a Send_error call before sending the complete logical record.</p> <p>System Action: The system returns this return code to the local TP on a Receive call that occurs after the TP receives the truncated logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)

Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)

Return Code	Value, Meaning, and Action
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocated_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
32	<p>Value: SVC_error_no_trunc</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation was in Send state, and the call did not truncate a logical record.</p> <p>System Action: The system returns this return code to the caller of the Receive service. The system does not return this return code to callers of the CPI Communications Send_Error call.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
34	<p>Value: SVC_error_trunc</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in send state, and the call truncated a logical record. Truncation occurs when a program begins sending a logical record and calls the Send_Error service before the complete record is sent.</p> <p>System Action: The system returns this return code when the local TP calls the Receive service to receive the truncated logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)

Return Code	Value, Meaning, and Action
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo)</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 31. Return Codes for the Receive_and_Wait Service (continued)	
Return Code	Value, Meaning, and Action
132	<p>Value: Deallocate_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

Transaction programs that call the Receive_and_Wait service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Request_to_Send

Equivalent to:

- LU 6.2 (MC_)Request_to_Send

- CPI Req_to_Send (CMRTS)

Notifies the partner program that the local program is requesting to enter Send state for the conversation. The conversation will be changed to Send state when the local program subsequently receives a send indication from the partner program.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBRTS(
    Conversation_id,
    Notify_type,
    Return_code
);
```

Figure 49. ATBRTS - LU 6.2 Request to Send

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Request_To_Send.

Return Codes

Valid return code values for the Return_code parameter are:

Table 32. Return Codes for the Request_to_Send Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 32. Return Codes for the Request_to_Send Service (continued)

Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The Sync_level is set to syncpt, and the TP is in the Backout-required state.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Request_to_Send service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Send_Data

Equivalent to:

- LU 6.2 (MC_)Send_Data
- CPI Send_Data (CMSEND)

Sends data to a partner program.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space, except for the <i>buffer</i> parameter.

Format

```
CALL ATBSEND(  
    Conversation_id,  
    Send_type,  
    Send_length,  
    Access_token,  
    Buffer,  
    Request_to_send_received,  
    Notify_type,  
    Return_code  
);
```

Figure 50. ATBSEND - LU 6.2 Send Data

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Send_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Send_type specifies what, if any, information is to be sent to the partner program in addition to the data supplied. Send_type also lets you combine operations (for example, Send_and_confirm) and save extra calls to APPC.

Valid values for this parameter are:

Value

Meaning

0

Buffer_data

Specifies that no additional information is to be sent to the partner program, and the data may be buffered until a sufficient quantity is accumulated.

1

Send_and_flush

Specifies that no additional information is to be sent to the partner program. However, the supplied data is sent immediately rather than buffered. This is functionally equivalent to a Send_data call with the Send_type parameter set to Buffer_data followed by a Flush call.

2

Send_and_confirm

Specifies that the supplied data is to be sent to the partner program immediately, along with a request for confirmation. This is functionally equivalent to a Send_data call with the Send_type parameter set to Buffer_data followed by a Confirm call.

3

Send_and_prepare_to_receive

Specifies that the supplied data is to be sent to the partner program immediately, along with send control of the conversation. This is functionally equivalent to a Send_data call with the Send_type parameter set to Buffer_data followed by a Prepare_to_receive call with the prepare_to_receive_type set to sync_level and the locks parameter set to short.

4

Send_and_deallocate

Specifies that the supplied data is to be sent to the partner program immediately, along with a deallocation notification. This is functionally equivalent to a Send_data call with the Send_type parameter set to Buffer_data followed by a Deallocate call with the deallocate_type set to sync_level.

Send_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Send_length specifies the length of the data to be sent. This data length is not related in any way to the length of the logical record. It is used only to determine the length of the data contained in the Buffer parameter.

If Send_length is zero, no data is sent for this call and the Buffer parameter is not significant.

Access_token

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Access_token specifies the Access List Entry Token (ALET) of the address space or data space in which the buffer resides for Send_Data calls.

APPC/MVS always uses access_token together with the address of the buffer to resolve addressing to the transaction program's data. To specify that the buffer address passed should not be ALET qualified, an Access_token value of zero should be supplied. APPC/MVS will then consider the buffer to reside in the primary address space of the caller.

The Access_token can:

- Represent an entry on the dispatchable unit access list (DU-AL)
- Represent an entry on the caller's primary address access list (PASN-AL), *only if the entry points to a SCOPE=COMMON data space.*

The Access_token cannot:

- Be the value 1 (which indicates "secondary ASID")
- Represent an entry on the caller's PASN-AL that *does not* point to a SCOPE=COMMON data space.

For more information about ALETs for SCOPE=COMMON data spaces, see [“Features of the MVS-Specific Services” on page 32.](#)

Buffer

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 0-2,147,483,647 bytes

Buffer specifies the data to be sent or received. This data can consist entirely of data (for mapped conversations) or logical records (for basic conversations).

Send_Data

If the data consists of logical records, each such record consists of a two-byte length field followed by a data field; the length of the data field can range from zero to 32,765 bytes. The length of the record includes the two-byte length field; therefore, logical-record length values of X'0000', X'0001', X'8000', and X'8001' are not valid.

Request_to_send_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Request_to_send_received specifies whether or not Request_to_send notification has been received.

Valid return values for this parameter are:

Value

Meaning

0

Request_to_send_not_received

indicates that Request_to_send notification has not been received

1

Request_to_send_received

the partner program has issued a Request_to_send, requesting the local program to enter Receive state.

If Return_code indicates Program_parameter_check or Program_state_check, a value is not returned in Request_to_send_received.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Send_Data.

Return Codes

The following table lists all of the possible return codes for Send_Data:

Table 33. Return Codes for the Send_Data Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>

Table 33. Return Codes for the Send_Data Service (continued)	
Return Code	Value, Meaning, and Action
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Ask the partner system programmer to provide a valid partner TP name. When requesting the allocate, specify the valid partner TP name.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 33. Return Codes for the Send_Data Service (continued)

Return Code	Value, Meaning, and Action
17	<p>Value: Deallocate_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 33. Return Codes for the Send_Data Service (continued)	
Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 33. Return Codes for the Send_Data Service (continued)

Return Code	Value, Meaning, and Action
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 33. Return Codes for the Send_Data Service (continued)

Return Code	Value, Meaning, and Action
132	<p>Value: Deallocated_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

Transaction programs that call the Send_Data service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Send_Error

Equivalent to:

- LU 6.2 (MC_)Send_Error

- CPI Send_Error (CMSERR)

Informs the partner program that the local program has detected an error.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBSERR(
    Conversation_id,
    Request_to_send_received,
    Notify_type,
    Error_Direction,
    Return_code
);
```

Figure 51. ATBSERR - LU 6.2 Send Error

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Request_to_send_received

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Request_to_send_received specifies whether or not Request_to_send notification has been received.

Valid return values for this parameter are:

Value

Meaning

0

Request_to_send_not_received

Indicates that Request_to_send notification has not been received.

Send_Error

1

Request_to_send_received

The partner program has issued a Request_to_send, requesting the local program to enter Receive state.

If Return_code indicates Program_parameter_check or Program_state_check, a value is not returned in Request_to_send_received.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Error_Direction

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Error_Direction specifies the direction of the data flow in which the program detected an error. This parameter is significant only if the Send_Error service is issued in Send_Pending state.

Valid values for this parameter are:

Value

Meaning

0

Receive_Error

Specifies that the program detected an error in the data it received from another program.

1

Send_Error

Specifies that the program detected an error while preparing to send data to the partner program.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See the following section for descriptions of return codes that can be returned to a caller of Send_Error.

Return Codes

If Send_Error is called in Send state, possible values for Return_code are:

Decimal Value

Meaning

0	OK
3	Conversation_type_mismatch
5	PIP_not_specified_correctly
6	Security_not_valid
8	Sync_lvl_not_supported_pgm
9	TPN_not_recognized
10	TP_not_available_no_retry
11	TP_not_available_retry
17	Deallocated_abend
20	Product_specific_error
22	Program_error_purging
24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_SVC
31	Deallocated_abend_timer

Send_Error

33

SVC_error_purging

100

Take_backout

130

Deallocated_abend_bo

131

Deallocated_abend_svc_bo (basic conversations only)

132

Deallocated_abend_timer_bo (basic conversations only)

133

Resource_failure_no_retry_bo

134

Resource_failure_retry_bo

Note: Return codes 100 through 134 are possible values only for conversations with a synchronization level of syncpt.

If Send_Error is called in Receive state, possible values for Return_code are:

Decimal Value

Meaning

0

OK

18

Deallocated_normal

20

Product_specific_error

24

Program_parameter_check

25

Program_state_check

26

Resource_failure_no_retry

27

Resource_failure_retry

133

Resource_failure_no_retry_bo

134

Resource_failure_retry_bo

135

Deallocated_normal_bo

Note: Return codes 133 through 135 are possible values only for conversations with a synchronization level of syncpt.

If Send_Error is called in Send-pending state, possible values for Return_code are:

Decimal Value

Meaning

0

OK

17

Deallocated_abend

20	Product_specific_error
22	Program_error_purging
24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry
30	Deallocated_abend_SVC
31	Deallocated_abend_timer
33	SVC_error_purging
100	Take_backout
130	Deallocated_abend_bo
131	Deallocated_abend_svc_bo (basic conversations only)
132	Deallocated_abend_timer_bo (basic conversations only)
133	Resource_failure_no_retry_bo
134	Resource_failure_retry_bo

Note: Return codes 100 through 134 are possible values only for conversations with a synchronization level of syncpt.

If Send_Error is called in Confirm, Sync_point, Sync_point_send, or Sync_point_deallocate state, possible values for Return_code are:

Decimal Value	Meaning
---------------	---------

0	OK
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check
26	Resource_failure_no_retry
27	Resource_failure_retry

133

Resource_failure_no_retry_bo

134

Resource_failure_retry_bo

Note: Return codes 133 and 134 are possible values only for conversations with a synchronization level of syncpt.

The following table describes *all* of the possible return codes for Send_Error:

Table 34. Return Codes for the Send_Error Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
5	<p>Value: PIP_not_specified_correctly</p> <p>Meaning: The partner LU rejected an allocate request. The partner TP defined one or more initialization parameter (PIP) variables, which APPC/MVS does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate. The system does not return this code to callers of the CPI Communications Allocate call.</p> <p>Application Programmer Response: Ask the partner system programmer to change the partner TP so it does not expect PIP data from the TP running on MVS.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 34. Return Codes for the Send_Error Service (continued)

Return Code	Value, Meaning, and Action
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the Allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocated_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocate_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 34. Return Codes for the Send_Error Service (continued)

Return Code	Value, Meaning, and Action
18	<p>Value: Deallocated_normal</p> <p>Meaning: A partner TP called the Deallocate service for a basic or mapped conversation. The request specified a Deallocate_type of Deallocate_sync_level or Deallocate_flush.</p> <p>System Action: The system returns this return code to the local TP when it calls a service while the conversation is in Receive state.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The local TP called an APPC service. One of the following errors occurred in one or more parameters specified on the call:</p> <ul style="list-style-type: none"> • An unauthorized caller specified a Notify_type of ECB. • The Sync_level field for the conversation was equal to sync_level_none. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 34. Return Codes for the Send_Error Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399. The conversation was in send state and the TP started, but the TP did not finish sending a logical record.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>

Table 34. Return Codes for the Send_Error Service (continued)	
Return Code	Value, Meaning, and Action
33	<p>Value: SVC_error_purging</p> <p>Meaning: A partner TP called the Send_Error service, and LU services on the partner LU specified a value of SVC for the type of call. The conversation for the partner TP was in Receive or Confirm state, and the call might have caused information to be purged.</p> <p>System Action: The system normally returns this code to the local TP after the system sends some information to the partner TP. However, the system can also return this code to the local TP before it sends any information.</p> <p>Application Programmer Response: See the application programmer response for return code six for this service.</p>
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 34. Return Codes for the Send_Error Service (continued)

Return Code	Value, Meaning, and Action
132	<p>Value: Deallocated_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
135	<p>Value: Deallocated_normal_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt.</p> <p>When the Send_Error call is issued in Receive state, incoming information is purged by the system. This purged information might include an abend deallocation notification from the partner program or system. The conversation is now in Reset state.</p> <p>System Action: /The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

TPs that call the Send_Error service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Set_Syncpt_Options

Equivalent to:

- LU 6.2 Set_Syncpt_Options
- (No CPI equivalent)

Changes the default system values that govern APPC/MVS processing during its participation in the two-phase commit protocol for resource recovery processing. The following options and their default values are in effect for each TP that issues Commit calls, for all protected conversations in which the TP participates, from the time the TP begins processing until Set_Syncpt_Options is issued to change them:

Option

System Default Value

Vote_Read_Only_Permitted

NO

Wait_For_Outcome

YES

Action_If_Problems

BACKOUT

The TP must specify valid values for all parameters on each invocation of Set_Syncpt_Options service.

For a call to Set_Syncpt_Options to complete successfully, TP resources must be associated with the calling address space. TP resources are associated with the calling address space after:

- A call to the CPI-C Initialize Conversation (CMINIT) service
- A call to the LU 6.2 Allocate service
- A call to the Register_for_Allocate service
- A call to the Define_Local_TP service

A program may successfully call Set_Syncpt_Options in the following situations as well:

- After an alternate transaction scheduler's XCF message user routine has received an Allocate TP request message for the TP
- After an alternate transaction scheduler receives and processes an Allocate TP request, and the TP begins running
- Before a call to either the Get_Conversation, or the CPI-C Accept_Conversation (CMACCP) service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller cannot hold any locks

Control parameters:	All parameters must be addressable by the caller and in the primary address space.
----------------------------	--

Format

```
CALL ATBSS04(
    Vote_Read_Only_Permitted,
    Wait_For_Outcome,
    Action_If_Problems,
    Reason_code,
    Return_code
);
```

Figure 52. ATBSS04 - LU 6.2 Set_Syncpt_Options

Parameters

Vote_Read_Only_Permitted

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Vote_Read_Only_Permitted specifies whether the local LU may vote read only in a syncpoint operation, if:

- The local LU has made no changes to protected resources, **and**
- None of the resource managers subordinate to the local LU, in the distributed transaction, have made any changes.

Valid values for this parameter are:

Value

Meaning

0

Specifies that the current value for this parameter should remain in effect (no change to the current setting).

1

NO

Specifies that voting read-only in a syncpoint operation is not allowed.

2

YES

Specifies that voting read-only in a syncpoint operation is allowed.

Wait_For_Outcome

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Wait_For_Outcome specifies whether the outcome of the syncpoint operation, at all subordinate resources in the distributed transaction, must be known before control is returned to the program. If a failure occurs and the TP specified Wait_For_Outcome with a value of 1 (NO), the system might return control from the Commit call with a return code value of rr_*_outcome_pending. This return code value indicates that the outcome at one or more distributed partners is unknown.

If the TP specified Wait_for_Outcome with a value of 2 (YES), the system does not return control until the outcome of the syncpoint operation at all subordinate resources is known.

Set_Syncpt_Options

Note: Even if the local TP specifies a value of 2 (YES), the system might return control before the outcome is fully known, if other programs in the distributed transaction specified Wait_for_Outcome with 1 (NO).

Valid values for this parameter are:

Value

Meaning

0

Specifies that the current value for this parameter should remain in effect (no change to the current setting).

1

NO

Specifies that the TP does not need to wait for the outcome of the syncpoint operation before regaining control.

2

YES

Specifies that the TP must wait for the outcome of the syncpoint operation before regaining control.

Action_If_Problems

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Action_If_Problems specifies the action taken by APPC/MVS when a unit of recovery associated with the TP is in the IN-DOUBT state and APPC/MVS receives an unrecognized message (a protocol violation) from the partner (syncpoint initiator) that is supposed to instruct it to commit or backout the unit of recovery.

Valid values for this parameter are:

Value

Meaning

0

Specifies that the current value for this parameter should remain in effect (no change to the current setting).

1

Commit

Specifies that the LU should instruct the syncpoint manager to heuristically commit the unit of recovery because the overall decision of the syncpoint initiator is unknown.

2

Backout

Specifies that the LU should instruct the syncpoint manager to heuristically backout the unit of recovery because the overall decision of the syncpoint initiator is unknown.

Reason_code

Returned parameter

- Type: Integer
- Length: 32 bits

Reason_code contains additional information about the result of the call when the return_code parameter contains a nonzero value.

Table 35 on page 251 lists the return and reason codes, their values, and meanings, associated with the Set_Syncpt_Options service return codes.

Table 35. Return and Reason Codes for Set_Syncpt_Options		
Return Code (Decimal)	Reason Code (Decimal)	Value, Meaning, and Action
0	—	<p>Value: atb_ok</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	All	<p>Value: atb_product_specific_error</p> <p>Meaning: The service was unsuccessful. APPC/MVS detected an environmental error during the processing of this callable service. See Table 12 on page 114 for the list of reason codes that accompany the product_specific_error return code.</p> <p>System Action: The system returns a product_specific_error (decimal 20) return code to the caller of the service. APPC/MVS writes a logrec data set record that describes the error. The system might request an SVC dump.</p> <p>Application Programmer Response: Contact the system programmer.</p> <p>System Programmer Response: Contact the IBM Support Center. Provide the logrec data set error record and the SVC dump (if one is available).</p>
24	All	<p>Value: atb_program_parameter_check</p> <p>Meaning: A user-supplied parameter was found to be in error. For example, a parameter contains characters not in the required character set.</p> <p>System Action: The system returns a program_parameter_check (decimal 24) return code to the caller.</p> <p>Application Programmer Response: See the value returned in the Reason_code parameter to determine the specific parameter that contained an invalid value. Supply a valid value and re-issue the service call.</p>
24	1	<p>Value: atb_invalid_vote_read_only</p> <p>Meaning: The specified Vote_read_only_permitted parameter does not contain a valid value.</p> <p>System Action: The system returns a program_parameter_check (decimal 24) return code to the caller.</p> <p>Application Programmer Response: Provide a valid value for the Vote_read_only_permitted parameter and re-issue the service call.</p>

Table 35. Return and Reason Codes for Set_Syncpt_Options (continued)		
Return Code (Decimal)	Reason Code (Decimal)	Value, Meaning, and Action
24	2	<p>Value: atb_invalid_wait_for_outcome</p> <p>Meaning: The specified Wait_for_outcome parameter does not contain a valid value.</p> <p>System Action: The system returns a program_parameter_check (decimal 24) return code to the caller.</p> <p>Application Programmer Response: Provide a valid value for the Wait_for_outcome parameter and re-issue the service call.</p>
24	3	<p>Value: atb_invalid_action_if_problems</p> <p>Meaning: The specified Action_if_problems parameter does not contain a valid value.</p> <p>System Action: The system returns a program_parameter_check (decimal 24) return code to the caller.</p> <p>Application Programmer Response: Provide a valid value for the Action_if_problems parameter and re-issue the service call.</p>
25	All	<p>Value: atb_program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid.</p> <p>System Action: The system returns a program_state_check (decimal 25) return code to the caller, and APPC/MVS might write a symptom record in the logrec data set, which provides further information about the program state check.</p> <p>Application Programmer Response: See the Application Programmer Response for the specific reason code that was returned on the service call.</p>
25	4	<p>Value: atb_extract_exit_not_specified</p> <p>Meaning: An APPC/MVS service was invoked in an address space that has more than one active TP. APPC/MVS could not associate the request with a TP because the transaction scheduler for the address space did not specify an extract exit.</p> <p>System Action: The system returns a program_state_check (decimal 25) return code to the caller.</p> <p>Application Programmer Response: Contact the system programmer.</p> <p>System Programmer Response: Contact the owner of the scheduler product. Ask the owner of the scheduler to ensure that the scheduler product specifies an extract exit. See the description of the Identify service in <i>z/OS MVS Data Areas</i> in the <i>z/OS Internet library</i> (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for information about how to establish an extract exit.</p>

Table 35. Return and Reason Codes for Set_Syncpt_Options (continued)

Return Code (Decimal)	Reason Code (Decimal)	Value, Meaning, and Action
25	5	<p>Value: atb_extract_exit_failed</p> <p>Meaning: A TP called an APPC/MVS conversation service in an address space where more than one TP was running. APPC/MVS called the transaction scheduler extract exit to identify the active TP. The exit returned a non-zero return code to APPC/MVS.</p> <p>System Action: The system returns a program_state_check (decimal 25) return code to the caller and APPC/MVS writes a symptom record in the logrec data set.</p> <p>Application Programmer Response: Contact the system programmer.</p> <p>System Programmer Response: See the symptom record in the logrec data set for a description of the error. Check the return code from the transaction scheduler extract exit in the scheduler extract control block (ATBSECB) in section 5 of the symptom record. The ATBSECB is in the first key-length-data structure in section 5. See <i>z/OS MVS Data Areas</i> in the <i>z/OS Internet library</i> (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the ATBSECB.</p>
25	6	<p>Value: atb_no_active_tp</p> <p>Meaning: The call for this service was performed prior to APPC/MVS allocating resources for the calling address space.</p> <p>System Action: The system returns a program_state_check (decimal 25) return code to the caller.</p> <p>Application Programmer Response: Issue the service call after issuing one of the following APPC/MVS services that will associate TP resources to the calling address space:</p> <ul style="list-style-type: none"> • CPI-C Initialize_Conversation (CMINIT) • LU 6.2 Allocate • Register_For_Allocates • Define_Local_TP <p>Ensure that all the conditions required for a call to this service are met. See “Set_Syncpt_Options” on page 248.</p>

Table 35. Return and Reason Codes for Set_Syncpt_Options (continued)		
Return Code (Decimal)	Reason Code (Decimal)	Value, Meaning, and Action
25	7	<p>Value: atb_service_error</p> <p>Meaning: A TP called an APPC/MVS conversation service. An internal error occurred in APPC/MVS processing.</p> <p>System Action: The system returns a program_state_check (decimal 25) return code to the caller and APPC/MVS writes a symptom record in the logrec data set. The system might request an SVC dump.</p> <p>Application Programmer Response: Contact the system programmer.</p> <p>System Programmer Response: Contact the IBM Support Center. Provide the logrec data set error record and the SVC dump (if one is available).</p>

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. The following table contains general descriptions of the possible return code values. See [Table 35 on page 251](#) for descriptions of possible reason codes.

Characteristics and Restrictions

Transaction programs that call the Set_Syncpt_Options service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note: If a TP calls Set_Syncpt_Options while a syncpoint operation is in progress for the current context under which the TP is running, the Set_Syncpt_Options request might not complete before APPC/MVS needs to refer to the syncpoint options during the syncpoint operation. In this case, the option values will not be changed in time for APPC/MVS to use the new values. Instead, APPC/MVS uses the option values in effect before the TP issued the call to Set_Syncpt_Options. Whenever possible, design APPC/MVS programs to issue the call to Set_Syncpt_Options before a syncpoint operation begins.

Set_TimeOut_Value

Sets the time limit in minutes and seconds that each subsequent APPC/MVS conversation call will wait for VTAM APPCCMD requests to complete. For more information, see [“Setting a Timeout Value for Potential Network Delays” on page 55](#).

The Set_Timeout_Value service can also be invoked to alter the previously set timeout_value.

For outbound transaction programs, the Set_TimeOut_Value service can be invoked at any time after the conversation is successfully established by the Allocate or CMINIT service.

For inbound transaction programs, the Set_TimeOut_Value service can be invoked at any time after successful completion of the Get_Conversation, Receive_Allocate, or CMACCP service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBSTO6(
    Conversation_id,
    Timeout_Value_Minutes,
    Timeout_Value_Seconds,
    Return_code
);
```

Figure 53. ATBSTO6 - Set_TimeOut_Value

Parameters

Conversation_ID

Supplied parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_ID specifies the conversation ID of the conversation for which you want to time VTAM APPCCMD requests issued during APPC/MVS conversation callable services. Specify the conversation_id that was returned from the Allocate, CMINIT, CMACCP, Get_Conversation, or Receive_Allocate call.

Timeout_Value_Minutes

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Value range: 0-1440 (decimal)

Specifies the time in minutes that all subsequent APPC/MVS conversation callable services will wait for VTAM APPCCMD requests to complete.

The maximum supported Timeout_Value_Minutes is 1440 minutes (24 hours). The total APPC timeout value is obtained from this parameter and the Timeout_Value_Seconds parameter. The two values are combined together to give the complete amount of time APPC should wait before timing out the conversation. If both Timeout_Value_Minutes and Timeout_Value_Seconds are zero, VTAM APPCCMD requests issued by subsequent APPC/MVS conversation callable services will not be timed.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Timeout_Value_Seconds

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Value range: 0-59 (decimal)

Specifies the time in seconds that all subsequent APPC/MVS conversation callable services will wait for VTAM APPCCMD requests to complete.

The maximum value supported for Timeout_Value_Seconds is 59 seconds. The total APPC timeout value is obtained from this parameter and the Timeout_Value_Minutes parameter. The two values are combined together to give the complete amount of time APPC should wait before timing out the conversation. If both Timeout_Value_Minutes and Timeout_Value_Seconds are zero, VTAM APPCCMD requests issued by subsequent APPC/MVS conversation callable services will not be timed.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. Possible values of Return_code are:

Return Code	Value, Meaning and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_Specific_Error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write the symptom records which describe the error to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_Parameter_Check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Set_Timeout_Value service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information

about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Chapter 9. APPC/MVS Advanced TP Callable Services

APPC/MVS provides a group of more advanced callable services that are specific to MVS and have no LU 6.2 or CPI equivalent.

Programs that use APPC/MVS advanced TP services must be link-edited with the APPC/MVS advanced TP services routine, ATBATP, and optionally, ATBCTS (if your program calls Reject_Conversation or Set_Conversation_Accounting_Information). The ATBATP and ATBCTS routines are shipped in SYS1.CSSLIB.

Advanced TP Callable Services with Multiple Call Names

The following table lists the advanced TP callable services that have more than one associated call name. This chapter describes the current versions of the calls, which are the preferred programming interfaces for these services. The previous versions are described in [Appendix E, “Previous Versions of APPC/MVS Callable Services,”](#) on page 425.

<i>Table 36. Advanced TP Callable Services with Multiple Call Names</i>			
Service Name	Previous Call Name	Current Call Name	Reference for Current Call
Cleanup_TP	ATBCMCTU	ATBCUC1	“Cleanup_TP” on page 263

Asynchronous_Manager

You can call the Asynchronous_Manager service either to determine whether there are any asynchronous APPC/MVS calls outstanding in an address space or to clean up an entire TP for which an asynchronous request is outstanding. Asynchronous APPC/MVS calls include any calls that have asynchronous processing specified on the Notify_Type parameter.

The Asynchronous_Manager service is particularly useful in TSO/E environments because TSO/E does not support the invocation of authorized commands or programs while an asynchronous APPC/MVS call is outstanding in the same address space. A program that needs to invoke an authorized command or program could first call the Asynchronous_Manager service to find out if there are any asynchronous calls outstanding in the address space and, if necessary, call Asynchronous_Manager again to clean up the TPs that submitted those calls. For example, you can call the Asynchronous_Manager service to clean up a TP in response to message IKJ56610I, which indicates attempted invocation of an authorized command or program while an asynchronous call is outstanding.

Note that the Asynchronous_Manager service only returns the number of asynchronous calls that are outstanding at the time the Asynchronous_Manager service is called; subsequent asynchronous calls are not reflected. Therefore, the query function of the Asynchronous_Manager service is most effective in an address space with no other tasks performing asynchronous APPC/MVS work.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit

ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBAMR1 (
    Function,
    Asynchronous_number,
    Return_Code
);
```

*Figure 54. ATBAMR1 - Asynchronous_Manager***Parameters****Function**

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Specifies the function to be performed. Valid values for this parameter are:

Value**Meaning****1**

Query - Asynchronous_Manager returns the number of outstanding asynchronous calls for the address space.

2

Cleanup - Asynchronous_Manager cleans up any TPs that have outstanding asynchronous calls. The cleaned up TPs receive a return code of Deallocated_abend. See [Appendix B, “Explanations of Return Codes for CPI Communications Services,” on page 391](#) for an explanation of this return code.

Asynchronous_Number

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Specifies the number of outstanding asynchronous requests in the address space. This field is not modified for the Cleanup function.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

The Asynchronous_Manager service may return one of the following values in the Return_code parameter:

Decimal Value Description

00

The Asynchronous_Manager service query function completed successfully. The Asynchronous_Number parameter contains a valid value.

The Asynchronous_Manager service completed successfully for the cleanup function. No TPs needed to be cleaned up. The Asynchronous_Number parameter is not updated.

04

The Asynchronous_Manager service completed successfully for the cleanup function. A TP was cleaned up. The Asynchronous_Number parameter is not updated.

16

An APPC/MVS service failure occurred.

20

The caller holds a lock or is disabled.

24

The function code is not valid.

32

The Asynchronous_Manager service was invoked for a scheduler address space.

36

The caller is in SRB mode.

44

APPC/MVS is not available.

48

APPC/MVS is ending.

Restrictions

Transaction programs that call the Asynchronous_Manager service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

When the Asynchronous_Manager has been called to clean up an entire TP for which an asynchronous request is outstanding and a protected conversation is associated with the TP to be cleaned up, APPC takes the following actions against the protected conversation:

Note: The outstanding asynchronous call does not have to be on a protected conversation for the following actions to be taken against a protected conversation associated with the TP to be cleaned up.

When a syncpoint operation IS in progress for the current UR for the context with which the protected conversation is associated, APPC/MVS does not immediately deallocate the conversation. The syncpoint operation is allowed to complete. As part of the syncpoint processing, the protected conversation might be deallocated, in which case no further cleanup is required for that conversation.

If the conversation was not deallocated, however, cleanup processing proceeds in the same manner as it does when a syncpoint operation IS NOT in progress at the time the Cleanup service is issued:

- The protected conversation is deallocated with TYPE(ABEND_SVC).
- The current UR is put into backout-required state.

Note: When a UR (unit of recovery) is in the backout-required state, no new APPC protected conversations may be allocated or accepted by a transaction program instance associated with the current UR nor may local protected resources associated with the current UR be committed. A Backout call should be issued to backout the local protected resources associated with the current UR. Once the current UR is backed out or the current context is ended, new APPC protected conversations may be associated with a new transaction program instance.

- If the protected conversation is an inbound conversation, the logical unit of work ID (LUWID) for the next UR is reset.
- The current UR and subsequent units of recovery for the context will not include the protected conversation being cleaned up by this service.

Accept_Test

The Accept_Test service specifies that the caller is ready to test the TP registered by a previous call to Register_Test. The Accept_Test service causes the caller to wait for the next inbound allocate request to arrive for that TP on behalf of that caller, and for the APPC test environment to be set up in the caller's address space.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBTEA1 ( TP_ID,  
              Return_Code  
            );
```

Figure 55. Invocation of the Accept_Test Callable Service

Parameters

TP_ID

Returned parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

The TP_ID is a token that represents the instance of the transaction program that was initiated for testing. To deallocate and clean up the conversation after the test is finished, supply this TP_ID on a call to the Cleanup_TP service.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Decimal Value Description

- 0** Accept_Test completed successfully.
- 4** The request was rejected because there was no Register_Test request active for the caller's address space.
- 8** APPC/MVS is not available.
- 12** The test request failed a RACF SECLABEL check.
- 16** APPC/MVS service failure.
- 20** The user was not authorized to execute the requested TP.
- 24** The test request was canceled by the Unregister_Test service.
- 28** There is an existing APPC conversation in the address space or conversation resources were not all cleaned up by previous conversations.

Restrictions

Transaction programs that call the Accept_Test service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Cleanup_TP

You can call the Cleanup_TP service from an unauthorized program to request that the APPC component clean up all conversation resources associated with a transaction program instance that is running in the caller's address space. Conversation resources include network resources, control blocks, and buffers which are used by the APPC component to manage the transaction program instance and its conversations.

Cleanup_TP is an unauthorized version of the Cleanup_TP scheduler service described in [z/OS MVS System Messages, Vol 3 \(ASB-BPX\)](#).

The primary use for Cleanup_TP is to clean up conversation resources left after testing a TP with the Register_Test and Accept_Test services.

The specified TP_ID is deleted from the system asynchronously as a result of this call, but cleanup processing also occurs asynchronously. Conversations with active APPC requests are not immediately deallocated. After the partner TP responds, APPC/MVS returns a deallocate condition and deallocates the conversation.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts

Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBCUC1 (TP_ID,
              Condition,
              Notify_Type,
              Return_Code
              );
```

Figure 56. ATBCUC1 - Cleanup_TP (Unauthorized Version)

Parameters

TP_ID

Supplied parameter

- Type: Character String
- Char Set: No restriction
- Length: 8 bytes

Specifies the transaction program instance which is to be cleaned up. All conversations owned by this transaction program instance are to be deallocated.

Condition

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Specifies the deallocation condition that has occurred. This field is used to determine the type of deallocate and sense code that is issued by the APPC component to the partner transaction program.

Valid values for this parameter are:

Value

Meaning

0

Normal

Specifies that the transaction program completed normally, even though it may have left active conversations. The APPC component deallocates all conversations in a proper state for normal deallocation with DEALLOCATE TYPE(SYNC_LEVEL). All conversations not in the proper state for a normal deallocation are deallocated with TYPE(ABEND_SVC).

1

System

Specifies that the transaction program ended abnormally. All active conversations are deallocated with TYPE(ABEND_SVC).

Note: If the value is not one of the values listed above, 0 (Normal) is used as the default.

Notify_Type

Supplied parameter

- Type: Structure
- Char Set: N/A

- Length: 4-8 bytes

Specifies the type of notification to be used to signal the TP that the TP_ID is deleted from the system.

APPC/MVS supports the following types of notification:

Type	Explanation
------	-------------

None	No notification is requested. When specified as a notification option for Cleanup_TP processing, the Cleanup_TP processing is performed synchronously, and control is returned to the caller when the TP_ID is deleted from the system. To specify this notification option, the caller must pass a four-byte structure containing a fullword binary zero.
ECB	Requests that the system perform notification by posting an ECB. The ECB to be posted is also specified on the Notify_Type parameter. When ECB notification is specified, Cleanup_TP processing is performed asynchronously, and control is returned to the caller before processing is complete. To specify this notification option, the caller must pass an eight-byte structure containing a fullword binary one (X'0000 0001'), followed by the address of a fullword area to be used as the ECB. APPC/MVS requires that the ECB reside in the home address space.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Cleanup_TP may return one of the following values in the return code parameter:

Decimal Value	Meaning
---------------	---------

0	Request accepted. All conversations owned by the transaction program instance will be cleaned up asynchronously.
4	No conversations exist to be cleaned up.
8	The TP_ID parameter specified a nonexistent transaction program instance.
12	The asynchronous request failed. Resubmit the request with a Notify_Type of None or report the problem to IBM.
20	Product specific error.
32	The requested service is not supported in the caller's environment. For example, this return code will be given if the caller invokes any of the transaction scheduler services when holding a lock.
44	APPC/MVS is not active.
48	APPC/MVS services failure.

Restrictions

- Transaction programs that call the Cleanup_TP service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT

FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

- Regardless of the condition parameter value specified for this service, APPC/MVS cleans up protected conversations differently, depending on whether a syncpoint operation is in progress. When a syncpoint operation **is** in progress for the current UR for the context with which the protected conversation is associated, APPC/MVS does not immediately deallocate the conversation. The syncpoint operation is allowed to complete. As part of the syncpoint processing, the protected conversation might be deallocated, in which case no further cleanup is required for that conversation.

If the conversation was not deallocated, however, cleanup processing proceeds in the same manner as it does when a syncpoint operation **is not** in progress at the time the Cleanup service is issued:

- The protected conversation is deallocated with TYPE(ABEND_SVC).
- The current UR is put into backout-required state.
- If the protected conversation is an inbound conversation, the logical unit of work ID (LUWID) for the next UR is reset.
- The current UR and subsequent units of recovery for the context will not include the protected conversation being cleaned up by this service.

Extract_Information

Extract_Information is a generalized service that you can use to extract detailed information about the conversations and scheduling of active APPC/MVS transaction programs. Extract_Information returns information that is not available from the Get_Attributes, Get_TP_Properties, or CPI Communications Extract_* calls.

Categories of Information

You can extract two categories of information: scheduling and conversation information, as specified by the Extract_code parameter.

Scheduling Information: The scheduling information that Extract_Information returns depends on the transaction scheduler under which the TP is running:

- When the transaction program is running under the APPC/MVS transaction scheduler, you can use Extract_Information to obtain the TP schedule type (standard or multi-trans) and additional information such as the transaction initiator class and times and dates when the TP was scheduled and initiated.
- When you request scheduling information for a TP running under another transaction scheduler, that scheduler's extract exit is called to return the requested information. The exit must provide whatever data is required by its published interface. Other transaction schedulers may provide their own format of data or give a return code indicating that no data is provided or that an error occurred. For information about providing an extract exit for a transaction scheduler, see *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

APPC Conversation Information: When you request APPC conversation information, you might be required to provide a value for the Qualifier_type or Qualifier_value parameter, or both, to indicate whether the request is for information about the caller's own conversation, or the conversation of another transaction program instance (TP_ID), or a specific conversation_id. The caller must be in supervisor state or PSW key 0-7 to:

- Request information about a specific TP_ID.
- Request information about a specific conversation_id that is not associated with the caller's address space.

When you specify extract_code X'0000' (summary conversation information), the service extracts the following conversation information for the transaction program indicated by Qualifier_type and Qualifier_value:

- Total number of conversations.

The total number of conversations associated with the TP_ID, currently active or deallocated. This includes the inbound conversation that might have started the TP, later inbounds that were processed (if the TP is a multi-trans), and all conversations started by issuing an Allocate call.

- Total number of allocated conversations (started by a CMALLC or Allocate call).
- Total number of Sends (CMSEND and Send_Data calls).
- Total amount of data (number of characters of data) sent from the program's send buffers.
- Total number of Receives (CMRCV, Receive_Immediate, and Receive_and_Wait calls).
- Total amount of data (number of characters of data) received by the program's receive buffers.
- Total number of currently active conversations (not deallocated or disconnected).

When you specify `extract_code X'0001'` (specific conversation information), the service extracts the conversation information shown in [“Contents of the Extract Buffer” on page 270](#) for the `conversation_id` indicated by `Qualifier_value`.

Requirements

Authorization:	Supervisor state or problem state, any PSW key (see “Characteristics and Restrictions” on page 270).
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space, except for the <i>buffer</i> parameter

Format

```
CALL ATBEXAI(
    Extract_code,
    Qualifier_type,
    Qualifier_value,
    Access_token,
    Buffer_length,
    Buffer,
    Return_code
);
```

Figure 57. ATBEXAI - Information Extract Service

Parameters

Extract_code

Supplied parameter

- Type: Integer
- Length: 32 bits

Specifies the code that identifies the unique information required by the calling program. For TPs running under the APPC/MVS transaction scheduler, specify one of the following values (right-justified):

- `Extract_code = X'1000'`: complete scheduling information
- `Extract_code = X'1001'`: scheduler name and schedule type only

Extract_Information

- Extract_code = X'0000': summary conversation information
- Extract_code = X'0001': specific conversation information.

For TPs processed by an APPC/MVS server, specify one of the following values:

- Extract_code = X'0000': summary conversation information
- Extract_code = X'0001': specific conversation information.

For TPs running under other transaction schedulers, you can specify an extract code value in the range X'1000' through X'1FFF' depending on which values the scheduler supports.

Qualifier_type

Supplied parameter

- Type: Integer
- Length: 32 bits

If you request scheduling information (Extract_code = X'1000' - X'1FFF'):

- The APPC/MVS transaction scheduler ignores any values for this parameter.
- Other transaction schedulers may accept values for this parameter; for TPs running under another transaction scheduler, any value you specify is passed as input in the EXTRACT_QUALTYPE field to the transaction scheduler extract exit.

If you request summary conversation information (Extract_code = X'0000'), the following values are acceptable:

- Qualifier_type = 0
Specifies conversation information about the caller. Qualifier_value should be ignored.
- Qualifier_type = 1
Specifies conversation information about a specified TP_ID. Qualifier_value must be the specified TP_ID. To use this value of Qualifier_type, the caller must be in supervisor state or PSW key 0-7.

If you request conversation-specific information (Extract_code = X'0001'), APPC/MVS ignores the Qualifier_type.

Qualifier_value

Supplied parameter

- Type: Character
- Char Set: No restriction
- Length: 8 bytes

The meaning of Qualifier_value is determined by the values you specify for Extract_code and Qualifier_type. Depending on those values, APPC/MVS treats Qualifier_value as follows:

- If you request complete scheduling information (Extract_code = X'1000'), Qualifier_value is ignored.
- If you request summary conversation information (Extract_code = X'0000'), the value in Qualifier_value is determined by the value you specify for Qualifier_type.
- If you request specific conversation information (Extract_code = X'0001'), Qualifier_value contains the identifier of the conversation (conversation_id) for which information is to be extracted. To request information about a conversation_id that is not associated with the caller's address space, the caller must be in supervisor state or PSW key 0-7.

When specified with a valid scheduler Extract_code for TPs running under another transaction scheduler, any value is passed as input in the EXTRACT_QUALTYPE field to the transaction scheduler extract exit.

Access_token

Supplied parameter

- Type: Integer

- Length: 32 bits

Access_token specifies the Access List Entry Token (ALET) of the address space or data space in which the buffer resides for Information_Extract calls.

APPC/MVS always uses Access_token together with the address of the buffer to resolve addressing to the transaction program's data. To specify that the buffer address passed should not be ALET qualified, an Access_token value of zero should be supplied. APPC/MVS will then consider the buffer to reside in the primary address space of the caller.

The Access_token can:

- Represent an entry on the dispatchable unit access list (DU-AL)
- Represent an entry on the caller's primary address access list (PASN-AL), *only if the entry points to a SCOPE=COMMON data space.*

The Access_token cannot:

- Be the value 1 (which indicates "secondary ASID")
- Represent an entry on the caller's PASN-AL that *does not* point to a SCOPE=COMMON data space.

For more information about ALETs for SCOPE=COMMON data spaces, see [“Features of the MVS-Specific Services” on page 32.](#)

Buffer_length

Supplied/Returned parameter

- Type: Integer
- Length: 32 bits

Specifies the length of the buffer where the information is to be returned.

Buffer_length will be updated to indicate the actual amount of the buffer used to return the data. This will be less than or equal to the Buffer_length that was supplied.

Buffer

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 0-32767 bytes

Specifies the buffer to contain extracted data. For information on the format of the extracted data that is returned, see [“Contents of the Extract Buffer” on page 270.](#)

Return_code

Returned parameter

Decimal Value

Description

0

Successful processing.

4

Buffer length too small. Only partial information is returned.

8

No information is returned for one of the following reasons:

- The caller is not a transaction program
- No transaction program was executing in the specified address space
- No scheduler extract exit was defined for the scheduler of this transaction program.

12

The Extract_code was incorrect or unsupported.

16

Extract_Information service failure.

20

The caller is disabled or holds a lock.

28

An incorrect version of ATB ATP was used to call Extract_Information.

36

The calling program specified incorrect or inconsistent parameters, or parameters it was not authorized to use.

44

APPC/MVS is not active.

48

The Qualifier_value was not valid.

Characteristics and Restrictions

1. The caller must be in supervisor state or PSW key 0-7 to do the following:
 - Request summary conversation information (Extract_code = X'0000') about a specific TP_id (Qualifier_type = 1)
 - Request information about a specific Conversation_id that is not associated with the caller's address space.
2. A program requiring information for a particular TP_id must extract it before the TP is terminated (Cleanup_Address_Space call is issued). Conversation information must be extracted before the conversation is deallocated. APPC conversation information is kept in real time and not logged by APPC/MVS.
3. The totals for conversation information are running totals (they are not reset after extraction). This service does not change or alter the contents of the data extracted.
4. Transaction programs that call the Extract_Information service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Contents of the Extract Buffer

Use the following mapping macros for the format and contents of the information extract buffer when Extract_Information is called with the extract codes supported by the APPC/MVS transaction scheduler. These mapping macros are documented in *z/OS MVS Data Areas* in the *z/OS Internet library* (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

For this type of extracted information:	Requested through extract code value:	Use this mapping macro:
Complete scheduling information	X'1000'	ATBEXSCH
Scheduler name and type only	X'1001'	ATBEXSCH
Summary conversation information	X'0000'	ATBEXCON
Specific conversation information	X'0001'	ATBEXCOS

For Summary Conversation Information (Extract Code X'0000')

There is no restriction on the largest number that the conversation totals can accumulate. To accommodate multi-trans programs that are started and stay in execution, each count in the "Total

amount of data sent” and “Total amount of data received” fields is returned as a normalized, long floating point number in the form X'eehhhhh hhhhhhhh', where:

- 'ee' consists of a '0' sign bit followed by a 7-bit characteristic (in excess 64 notation)
- 'hhhhh hhhhhh' is a normalized hexadecimal fraction in which:
 - 'k' represents a hexadecimal digit in the range 1-F (except when the floating point value is a “true zero”), and
 - Each 'h' represents a hexadecimal digit in the range 0-F.

When the hexadecimal value is 0, a “true zero” (X'00000000 00000000') is returned as the floating point representation.

Some examples of typical values returned are:

Packed-decimal representation	64-bit binary representation	Normalized Long FP representation
000000000000000C	0000000000000000	0000000000000000
000000000001000C	000000000000003E8	433E800000000000
000000000002048C	00000000000000800	4380000000000000
999999999999999C	00038D7EA4C67FFF	4D38D7EA4C67FFF0

For Specific Conversation Information (Extract Code X'0001')

Some of the information returned in the extract buffer is:

- EXCOS_PLU_LOCATION, which indicates whether the partner LU resides on this system (local) or another system in the network (remote). The possible values are:
 - 0**
APPC/MVS could not yet determine the location of the partner LU
 - 1**
Remote
 - 2**
Local
- EXCOS_CONV_KIND, which, for inbound conversations, indicates whether the conversation was processed by an APPC/MVS server or not (probably because a transaction scheduler processed the inbound conversation). For outbound conversations, this field always contains a zero. The possible values are:
 - 0**
Not processed by an APPC/MVS server
 - 1**
Processed by an APPC/MVS server.
- EXCOS_SCHED_NAME, which is the scheduler name. This field contains blanks for conversations not processed by a transaction scheduler, such as:
 - Inbound conversations processed by an APPC/MVS server
 - Outbound conversations from address spaces not associated with a transaction scheduler (such as a TSO/E user or a batch job).
- EXCOS_TP_NAME, which is the name of the partner TP. If the conversation is inbound, this field contains blanks.
- EXCOS_LOCAL_TP_NAME, which is the name of the local TP. If the conversation is outbound, this field contains the name of the program that initiated the conversation (through the Allocate service). If the conversation is inbound, this field contains the name of the program that was attached on this LU because of an Allocate call.

- EXCOS_CONV_START_TIME:
 - For **inbound conversations**, this field contains the date and time APPC/MVS routed the conversation to an address space for subsequent processing. For conversations processed by APPC/MVS servers, this is when the server received the conversation from an allocate queue (through the Receive_Allocate service). For scheduled conversations, this is when the transaction scheduler directed the allocate request to an initiator address space for processing.
 - For **outbound requests**, this field contains the date and time the local program called the Allocate service to initiate a conversation.

This information appears in the format provided by the STORE CLOCK (STCK) assembler instruction.

- EXCOS_LAST_SERVICE_RETURN_CODE, which is the last return code received from a callable service during this conversation. If the return code indicated a product-specific error (decimal 20), the next field in this buffer contains the reason code for the error.
- EXCOS_URID, which is the unit of recovery identifier for a protected conversation (conversation with a synchronization level of syncpt).

Get_Transaction

When APPC/MVS transaction programs are assigned a TP schedule type of multi-trans in the TP profile, those TPs can use the Get_Transaction service to hold consecutive conversations with multiple partner programs, without having to be terminated and reinitialized for each conversation.

TPs call Get_Transaction apart from the actual APPC conversation, in a part of the TP called a **multi-trans shell**. The multi-trans shell gets control first, during initialization, and issue Get_Transaction when the program is ready to handle the first incoming conversation request. Before issuing Get_Transaction to request consecutive conversations, the multi-trans shell is responsible for doing whatever cleanup is necessary to ensure that consecutive users maintain their data integrity and are isolated from one another and from resources used exclusively by the shell.

When APPC/MVS initiates a multi-trans TP, the TP's initial execution environment is set by a generic userid specified in the TP profile. When a conversation begins, the environment of the multi-trans TP changes to reflect the calling partner's security, accounting, and distribution characteristics. The environment is thus "personalized" for each consecutive caller that receives control from a subsequent Get_Transaction request. The multi-trans TP returns to its generic execution environment at termination or by issuing a Return_Transaction call from its multi-trans shell.

In response to each Get_Transaction call, the APPC/MVS transaction scheduler looks for the next incoming allocate request for the program. When it finds a request, APPC/MVS returns control from the Get_Transaction service with a return code of zero. The multi-trans shell then passes control to the part of the program that holds the actual conversation. Each Get_Transaction call deallocates any conversation received in response to a previous Get_Transaction call with a deallocate Type(Abend). For multi-trans TPs processing protected conversations, APPC/MVS causes all protected resources updated since the last commit or backout to be backed out if the multi-trans TP did not issue a Backout or Commit call prior to calling Get_Transaction to retrieve the next incoming allocate request.

For examples of using the Get_Transaction service with a multi-trans shell, see [“Examples of Multi-Trans Scheduling”](#) on page 62.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)

Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGTRN(
    Return_code
);
```

Figure 58. ATBGTRN - Obtaining the Next Transaction

Parameters

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program.

Valid return values for this parameter are:

Decimal Value

Description

0

An inbound transaction was obtained and is ready to run.

8

The Get_Transaction request waited for an interval based on installation response time goals and current demand for initiators in this class, up to a maximum of 5 minutes. No more work was available. The caller can call the Get_Transaction service again to wait for more work to arrive. The caller's execution environment is the environment that was set by the generic userid specified in the caller's TP profile.

12

The APPC/MVS transaction scheduler was not active. The caller *cannot* call the Get_Transaction service again to wait for more work to arrive.

16

Because of conditions in the TP or the TP's profile, no work was available. The calling environment is not valid because:

- The TP is not scheduled as multi-trans
- The TP was not running under an APPC/MVS transaction scheduler initiator
- The calling program was in cross memory mode or SRB mode when it called the Get_Transaction service.

The caller's execution environment remains unchanged from when the TP called the Get_Transaction service.

Verify that the TP was running under the APPC transaction scheduler, in task mode. Verify that the TP's profile specified a TPSCHED_TYPE of MULTI_TRANS.

20

System error.

24

A previous call to the Get_Transaction or Return_Transaction service is still in progress in the same address space.

The caller's security and job control language (JCL) environment remain unchanged when the Get_Transaction service is called. However, the security and JCL environment may change when the previous Get_Transaction or Return_Transaction call ends.

28

The operator entered one or more commands (or the system administrator entered a configuration definition) that caused no more work to be available. The caller cannot call the Get_Transaction service again to wait for more work to arrive. The TP should terminate. The caller's execution environment is the environment that was set by the generic userid specified in the caller's TP profile.

Note: The system can return this code in the following situations:

- The system operator entered a SET command that executed a CLASSDEL statement in an ASCHPMxx parmlib member, which specified that an APPC scheduler class is to be deleted. The scheduler class is running the multi-trans TP; the multi-trans TP has not called Get_Transaction service.
- The APPC scheduler forced the multi-trans TP to terminate so the APPC scheduler can run other work in the same APPC scheduler class.
- The system administrator updated the profile for the multi-trans TP while the multi-trans TP was either running or waiting for work.

Validate that the CLASS statement in the TP profile for the multi-trans TP is correct. If no errors are found, validate that no other TP profile has the same class name specified on the CLASS statement.

Restrictions

1. Transaction programs that call the Get_Transaction service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).
2. When a multi-trans TP creates subtasks, only one task can call Get_Transaction at a time. This is because each call to Get_Transaction changes the caller's execution environment until the subtask either terminates or calls Return_Transaction and receives a return code of zero.
3. When a multi-trans TP creates a new jobstep task or subtask, the new jobstep task or subtask (and all of its subtasks) cannot call Get_Transaction.
4. A multi-trans TP cannot call Get_Transaction while a previous call to Get_Transaction is still in progress in the same address space.

Register_Test

The Register_Test service requests that a TP be initiated for testing purposes in the caller's address space instead of in an APPC initiator. The service directs APPC/MVS to initiate the TP in the caller's address space when the next inbound allocate request for that TP, on behalf of that caller, arrives. The caller must set the appropriate environment for the TP, and must call the Accept_Test service when it is ready to receive and run the TP. The caller can include its own interactive debugging facilities to help test and diagnose the TP.

This service does not support multi-trans TPs.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
-----------------------	--

Dispatchable unit mode:	Task mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBTER1 ( TP_Name_Length,
               TP_Name,
               LU_Name_Length,
               LU_Name,
               Return_Code
             );
```

Figure 59. Invocation of the Register_Test Callable Service

Parameters

TP_Name_Length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Specifies the length of the data contained in the TP_Name parameter.

TP_Name

Supplied parameter

- Type: Character
- Char Set: 00640
- Length: 1-64 bytes

The name of the TP to be tested.

LU_Name_Length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Specifies the length of the data contained in the LU_Name parameter.

LU_Name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Specifies the LU where the TP to be registered resides. To specify the base LU for the transaction scheduler associated with the caller, supply all blanks. The LU must be associated with a transaction scheduler (that is, you cannot specify a NOSCHED LU).

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Decimal Value

Description

0

Register_Test completed successfully.

4

The request was rejected because a previous test request is active for the caller's address space. Only one test request is permitted per address space at any given time.

8

APPC/MVS is not available.

12

The TP name was either not specified or was invalid.

16

APPC/MVS service failure.

20

The user was not authorized to modify or create a user-level TP profile.

24

The LU name was not valid.

28

There is an existing APPC conversation in the address space or conversation resources were not all cleaned up by previous conversations.

32

There was an error in creating a user level TP profile.

36

APPC test enablement services are unavailable.

40

The LU specified was undefined or was not associated with the scheduler of this address space, or the specified LU name contains characters that are not defined to APPC/MVS.

44

LU_name was blank to request the base LU, but no base LU is defined.

48

An error occurred while Register_Test was trying to add or modify a TP profile for testing purposes.

50

The specified LU (or the default base LU, if no LU name was specified) was not associated with any scheduler.

Restrictions

Transaction programs that call the Register_Test service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Reject_Conversation

The Reject_Conversation service rejects an inbound conversation.

A program can call this service to avoid processing a particular inbound conversation. The caller must supply an appropriate sense code (as an input parameter) to indicate the reason the conversation was rejected. APPC/MVS resolves the sense code to a return code that it passes to the partner transaction.

A program cannot reject a conversation if there has been any communication activity performed on it. After the program has obtained the conversation id (through the Get_Conversation, Receive_Allocate, or Accept_Conversation service), only the following conversation services may be called before calling Reject_Conversation:

- Get_Type (ATBGETT)
- Get_Attributes (ATBGETA, ATBGTA2)
- Extract_Conversation_Type (CMECT)
- Extract_Mode_Name (CMEMN)
- Extract_Sync_Level (CMESL)
- Extract_Partner_LU_Name (CMEPLN).

If a program attempts to reject a conversation that has had communication activity, the Reject_Conversation service returns a return code of atbcts_request_unsuccessful, and a reason code of atbcts_not_first_conv_call.

For protected conversations:

- The Deallocate_sense_code is ignored when a syncpt conversation is rejected. A deallocated_abend_svc sense code (X'08640001') is used instead.
- The current UR is put into backout-required state.
- The current UR and subsequent units of recovery for the context will not include the protected conversation being rejected by this service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBRJC2(
    Notify_type,
    Conversation_id,
    Deallocate_sense_code,
    Reason_code,
    Return_code
);
```

Figure 60. ATBRJC2 - Reject_Conversation

Parameters

Notify_type

Supplied parameter

- Type: Structure
- Length: 4-8 bytes

Specifies the type of processing and notification requested for this service (synchronous or asynchronous). The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeroes.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service. The reason code, if any, is set in the caller's Reason_code parameter.

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Deallocate_sense_code

Supplied parameter

- Type: Integer
- Length: 4 bytes

Deallocate_sense_code specifies the sense code to be sent (as a return code) to the partner transaction program.

The sense code can be coded as an explicit hexadecimal value or as a symbolic. Valid values for this parameter are:

Value (hex)

Meaning

X'084B6031'

Atbcts_TP_Not_Available_Retry

Specifies that the conversation should be abnormally ended with an indication that the requested TP is not available. The partner may attempt to retry the request.

X'084C0000'

Atbcts_TP_Not_Avail_No_Retry

Specifies that the conversation should be abnormally ended with an indication that the requested TP is not available. The partner should not attempt to retry the request.

X'10086021'

Atbcts_TPN_Not_Recognized

Specifies that the conversation should be abnormally ended with an indication that the requested TP name is not recognized.

X'080F6051'

Atbcts_Security_Not_Valid

Specifies that the conversation should be abnormally ended with an indication that a security violation was detected.

X'10086041'

Atbcts_Sync_Lvl_Not_Spprtd_Pgm

Specifies that the conversation should be abnormally ended with an indication that the specified synchronization level is not supported.

X'10086034'

Atbcts_Conv_type_mismatch

Specifies that the conversation should be abnormally ended with an indication that the conversation type is not supported.

Reason_code

Returned parameter

- Type: Integer
- Length: 32 bits

Reason_code contains additional information about the result of the call when the return_code parameter contains a nonzero value other than atbcts_appc_not_available.

[Table 37 on page 279](#) lists the valid reason codes.

Return_code

Returned parameter

- Type: Integer
- Length: 32 bits

Return_code specifies the result of the call. If the return_code parameter contains zero or 64 (decimal), there is no reason code. For other return codes, check the reason_code parameter for additional information about the result of the call.

The following table lists the possible return and reason codes, their symbolic equates, and their meanings, for the Reject_Conversation service.

<i>Table 37. Return and Reason Codes for Reject_Conversation</i>		
Return Code (Decimal)	Reason Code (Decimal)	Symbolic and Meaning
0	—	atbcts_ok The service completed as requested.
8	All	atbcts_parameter_error A user-supplied parameter was found to be in error. For example, a parameter contains characters not in the required character set. See the reason_code parameter to determine which parameter is in error.
8	18	atbcts_inval_notify_type The specified notify type is not valid.

Table 37. Return and Reason Codes for Reject_Conversation (continued)		
Return Code (Decimal)	Reason Code (Decimal)	Symbolic and Meaning
8	22	atbcts_inval_conversation_id The specified conversation identifier does not represent an active conversation for this address space.
8	23	atbcts_inval_sense_code The specified sense code is not valid or not supported.
16	All	atbcts_request_unsuccessful The service was unsuccessful. The cause is most likely a parameter error other than a syntax error, or an environmental error. For example, a syntactically valid LU name was specified, but the LU is not defined to APPC/MVS. An example of an environmental error is that the caller called the service while holding locks. See the Reason_code parameter for the specific cause of the error, and to determine whether the error can be corrected and the service reissued.
16	8	atbcts_cannot_hold_locks The caller held one or more locks when calling the service.
16	24	atbcts_not_first_conv_call Reject_Conversation was called for a conversation that has already had a communication service issued on it.
16	25	atbcts_not_inbound_conv The specified conversation is not an inbound conversation.
16	40	atbcts_conv_inaccessible The specified conversation is currently in use by another process.
32	All	atbcts_service_failure APPC/MVS service failure. Record the return and reason code, and give them to your systems programmer, who should contact the appropriate IBM support personnel.
32	16	atbcts_appc_service_failure The service failed because of an APPC failure. APPC provides symptom records for this type of error. For more information, see the appendix that explains return and reason codes in z/OS MVS Programming: Writing Servers for APPC/MVS .
64	—	atbcts_appc_not_available APPC/MVS is not currently active. Call the service again after APPC is available.

Restrictions

Transaction programs that call the Reject_Conversation service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information

about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Return_Transaction

For a program that was scheduled with a schedule type of multi-trans, Return_Transaction returns the transaction program execution environment to the generic shell environment that was established when the transaction program was initialized. Return_Transaction deallocates any current conversation from a previous Get_Transaction call with a deallocate Type(Abend). For multi-trans TPs processing protected conversations, APPC/MVS causes all protected resources updated since the last commit or backout to be backed out if the multi-trans TP did not issue a Backout or Commit call prior to calling Return_Transaction for the next incoming allocate request.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBTRTN(
    Return_code
);
```

Figure 61. ATBTRTN - Restoring the Generic Environment

Parameters

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

Valid return values for this parameter are:

Decimal Value

Description

0

Generic environment restored.

4

Unable to restore generic environment.

12

The APPC transaction scheduler was not active.

16

The calling environment is not valid because:

- The TP is not scheduled as multi-trans
- The TP was not running under the APPC/MVS transaction scheduler address space
- The calling program was in cross memory mode or SRB mode when it called the Return_Transaction service.

The caller's security and job control language (JCL) environment remain unchanged when the Return_Transaction service is called.

20

System error.

24

A previous call to the Get_Transaction or Return_Transaction service from the same address space is still in progress.

The caller's security and job control language (JCL) environment remain unchanged when the Return_Transaction service is called. However, the security and JCL environment may change when the previous Get_Transaction or Return_Transaction call ends.

Restrictions

Transaction programs that call the Return_Transaction service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Set_Conversation_Accounting_Information

Places up to 255 bytes of user-defined data to the accounting record for the specified conversation (SMF record type 33, subtype 2). The user-defined data, and its length, is also placed in the buffer that the Extract_Information service returns.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBSCA2(
    Notify_type,
    Conversation_id,
    User_accounting_data_length,
    User_accounting_data,
    Reason_code,
    Return_code
);
```

Figure 62. ATBSCA2 - Set_Conversation_Accounting_Information

Parameters

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service. The reason code, if any, is set in the caller's Reason_code parameter.

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

User_accounting_data_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Range: 0-255

User_accounting_data_length specifies the length of data contained in the User_accounting_data parameter.

If zero is specified for this parameter, any previously written user accounting data is removed from this conversation's accounting record.

User_accounting_data

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 0-255 bytes

User_accounting_data specifies user-defined data to be placed in the SMF accounting record for this conversation.

Reason_code

Returned parameter

- Type: Integer
- Length: 32 bits

Reason_code contains additional information about the result of the call when the return_code parameter contains a nonzero value other than atbcts_appc_not_available.

Table 38 on page 284 lists the valid reason codes.

Return_code

Returned parameter

- Type: Integer
- Length: 32 bits

Return_code specifies the result of the call. If the return_code parameter contains zero or 64 (decimal), there is no reason code. For other return codes, check the reason_code parameter for additional information about the result of the call.

The following table lists the possible return and reason codes, their symbolic equates, and their meanings, for the Set_Conversation_Accounting_Information service.

<i>Table 38. Return and Reason Codes for Set_Conversation_Accounting_Information</i>		
Return Code (Decimal)	Reason Code (Decimal)	Symbolic and Meaning
0	—	atbcts_ok The service completed as requested.
8	All	atbcts_parameter_error A user-supplied parameter was found to be in error. For example, a parameter contains characters not in the required character set. See the reason_code parameter to determine which parameter is in error.
8	18	atbcts_inval_notify_type The specified notify type is not valid.
8	22	atbcts_inval_conversation_id The specified conversation identifier does not represent an active conversation for this address space.
8	35	atbcts_inval_acct_data_length The specified accounting data field length is outside the allowable range.

Table 38. Return and Reason Codes for Set_Conversation_Accounting_Information (continued)

Return Code (Decimal)	Reason Code (Decimal)	Symbolic and Meaning
16	All	atbcts_request_unsuccessful The service was unsuccessful. The cause is most likely a parameter error other than a syntax error, or an environmental error. For example, a syntactically valid LU name was specified, but the LU is not defined to APPC/MVS. An example of an environmental error is that the caller called the service while holding locks. See the Reason_code parameter for the specific cause of the error, and to determine whether the error can be corrected and the service reissued.
16	8	atbcts_cannot_hold_locks The caller held one or more locks when calling the service.
16	40	atbcts_conv_inaccessible The specified conversation is currently in use by another process.
32	All	atbcts_service_failure APPC/MVS service failure. Record the return and reason code, and give them to your systems programmer, who should contact the appropriate IBM support personnel.
32	16	atbcts_appc_service_failure The service failed because of an APPC failure. APPC provides symptom records for this type of error. For more information, see the appendix that explains return and reason codes in z/OS MVS Programming: Writing Servers for APPC/MVS .
64	—	atbcts_appc_not_available APPC/MVS is not currently active. Call the service again after APPC is available.

Restrictions

Transaction programs that call the Set_Conversation_Accounting_Information service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

Unregister_Test

The Unregister_Test service cancels an outstanding TP test request before the test begins. You can call this service after calling Register_Test or Accept_Test. This service is useful if you change your mind about performing the test or are unable to initiate an inbound allocate request for the TP.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode

Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBTEU1 ( Return_To_Program,
               ASCB_address,
               Return_code
             );
```

Figure 63. Invocation of the Unregister_Test Callable Service

Parameters

Return_to_program

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Specifies whether to return control to the program that called Register_Test. It has two possible values:

Value

Description

0

No - specifies that control not be returned to the program that called Register_Test.

1

Yes - specifies that control be returned to the program that called Register_Test.

ASCB_address

Supplied parameter

- Type: Pointer
- Char Set: N/A
- Length: 4 bytes

Specifies the pointer to the address space control block (ASCB) that represents the address space for which the Unregister_Test service should cancel the pending test request. Only programs with PSW key 0-7 can use this parameter. Problem programs (key 8) can only cancel test requests for their own address spaces and should pass a null pointer (X'00000000') or a pointer to their own address space.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Possible values from the Unregister_Test service are:

Decimal Value Description

0	Unregister_Test service completed successfully.
4	No active test request was found for the address space.
8	APPC/MVS is not available.
12	Problem state program supplied the address of an ASCB other than its own.
16	APPC/MVS service failure.
20	Value of Return_to_program parameter was not valid.

Restrictions

Transaction programs that call the Unregister_Test service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Version_Service

Version_Service returns the highest version of the APPC/MVS callable services that is currently available on the system. Available services are those with a version number that is equal to or between the version number where the service was introduced and the version number returned from the Version_Service.

Example

The following example explains the versioning of APPC/MVS callable services:

- Assume that the ATBxxx1 service was introduced in release A (the first release using this versioning scheme). The invocation of ATBxxx1 expects an Unidentify_type and a return code parameter.
- In the next release (release B) a new parameter is added to form the ATBxxx2 service. The invocation of the new service expects only a return code parameter.
- In the following release (release C), a new parameter is added to the ATBxxx2 service to form the ATBxxx3 service. No changes are made to the ATBxxx1 and ATBxxx2 services.

The following table shows how a call to the Version_Service returns callable service version numbers for the Allocate service:

Table 39. Example Values Returned by the Version_Service		
Current Version	Value Returned	Services Available
0 (MVS/ESA SP 4.2.0)	N/A (See note 1)	ATBALLOC
1 (MVS/ESA SP 4.2.2)	1	ATBALLOC
2 (MVS/ESA SP 4.3.0)	2	ATBALLOC, ATBALC2
3 (MVS/ESA SP 5.1.0)	3	ATBALLOC, ATBALC2
4 (OS/390 V1R3)	4 or 5 (See Note 2)	ATBALLOC, ATBALC2, ATBALC5
5 (OS/390 V2R8)	5	ATBALLOC, ATBALC2, ATBALC5

Table 39. Example Values Returned by the Version_Service (continued)

Current Version	Value Returned	Services Available
6 (z/OS V1R7)	6	ATBALLOC, ATBALC2, ATBALC5, ATBALC6

Note:

1. The Version_Service was not available in MVS/ESA SP 4.2.0.
2. If APAR OW33764 is installed, then the Version_Service will return a value of 5. Otherwise, a value of 4 will be returned.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Any
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.

Format

```
CALL ATBVERS(
    Callable_Service_Version_Number,
    Return_Code,
);
```

Figure 64. ATBVERS - Callable Service Version Service

Parameters**Callable_Service_Version_Number**

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

The highest version of all APPC/MVS callable services currently available on the system.

Return_Code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 4 bytes

Valid values for this parameter are:

Decimal Value	Description
---------------	-------------

0

The Version_Service successfully returned the version number.

48

A service failure occurred.

Restrictions

Transaction programs that call Version_Service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [*z/OS MVS Programming: Authorized Assembler Services Guide*](#).

Chapter 10. API Trace Facility Messages

This chapter complements the application programming interface (API) trace information presented in Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77. Messages for the API trace facility:

- Report ATBTRACE REXX exec syntax or processing errors
- Report the status of tracing activity
- Illustrate conversation data written to the trace data set.

These messages are either returned to the issuer of the ATBTRACE REXX exec, or written to the trace data set. Additional messages related to the API trace facility appear on the operator console; these messages are documented in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

**ATB60001I ATBTRACE SYNTAX ERROR:
UNEXPECTED TEXT "text" FOUND
INSTEAD OF START, STOP, OR LIST**

Explanation

On an ATBTRACE request, the system found text where it expected to find a valid ATBTRACE function. Valid functions are START, STOP, and LIST.

In the message text:

text

The text found on the ATBTRACE request where START, STOP, or LIST was expected.

System action

The system rejects the request.

Programmer response

Correct the syntax of the ATBTRACE request by specifying START, STOP, or LIST, and re-issue the request.

Detecting Module:

ATBVSTM, ATBTRACE exec

System action

The system rejects the request.

Programmer response

Correct the syntax of the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60003I ATBTRACE SYNTAX ERROR:
PARENTHESIS IS MISSING AFTER
TEXT "text"**

Explanation

On an ATBTRACE request, the system did not find an opening or closing parenthesis before or after a keyword value.

In the message text:

text

The keyword value that was found without either an opening or closing parenthesis.

System action

The system rejects the request.

Programmer response

Correct the syntax of the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60002I ATBTRACE SYNTAX ERROR:
UNEXPECTED TEXT "text" FOUND
INSTEAD OF A VALID KEYWORD**

Explanation

On an ATBTRACE request, the system found text where it expected to find a valid keyword.

In the message text:

text

The text found on the ATBTRACE request where a keyword was expected.

**ATB60004I ATBTRACE SYNTAX ERROR: THE
LU KEYWORD AND VALUE ARE
REQUIRED FOR A START REQUEST**

Explanation

On an ATBTRACE request, the system could not find the LU keyword. Both the keyword and its value, an LU name, are required for a START request.

System action

The system rejects the request.

Programmer response

Make sure the ATBTRACE request contains both the LU keyword and value, and re-issue the request.

Detecting Module:

ATBTRACE exec

Explanation

On an ATBTRACE request, the system could not find the DATASET keyword (or the accepted keyword abbreviations DSNAME, DA, DSN, or DS). Both the keyword and its value, a data set name, are required for a START or STOP request.

System action

The system rejects the request.

Programmer response

Make sure the ATBTRACE request contains both the DATASET keyword (or accepted abbreviation) and value, and re-issue the request.

Detecting Module:

ATBTRACE exec

**ATB60005I ATBTRACE SYNTAX ERROR: THE
TP KEYWORD AND VALUE ARE
REQUIRED FOR A START REQUEST**

Explanation

On an ATBTRACE request, the system could not find the TP keyword. Both the keyword and its value, a transaction program (TP) name, are required for a START request.

System action

The system rejects the request.

Programmer response

Make sure the ATBTRACE request contains both the TP keyword and value, and re-issue the request.

Detecting Module:

ATBTRACE exec

**ATB60007I ATBTRACE SYNTAX ERROR: THE
SYMDEST KEYWORD IS MUTUALLY
EXCLUSIVE WITH THE LU AND TP
KEYWORDS**

Explanation

On an ATBTRACE request, the system found the SYMDEST keyword specified with the LU or TP keyword, or both. You may specify either the SYMDEST keyword **or** the combination of both LU and TP.

System action

The system rejects the request.

Programmer response

Correct the ATBTRACE request to specify either SYMDEST **or** LU with TP, and re-issue the request.

Detecting Module:

ATBTRACE exec

**ATB60006I ATBTRACE SYNTAX ERROR: THE
DATASET KEYWORD AND VALUE
ARE REQUIRED FOR A START OR
STOP REQUEST**

**ATB60008I ATBTRACE SYNTAX ERROR:
THE USERID AND SECNONE
KEYWORDS ARE MUTUALLY
EXCLUSIVE**

Explanation

On an ATBTRACE request, the system found both the USERID and SECNONE keywords specified. You may specify either USERID **or** SECNONE, but not both.

System action

The system rejects the request.

Programmer response

Correct the ATBTRACE request to specify either USERID **or** SECNONE, and re-issue the request.

Detecting Module:

ATBTRACE exec

**ATB60009I ATBTRACE SYNTAX ERROR: THE
LU KEYWORD VALUE IS MISSING
OR TOO LONG**

Explanation

On an ATBTRACE request, the LU keyword value was either missing or too long. A network LU name cannot be greater than 8 characters. The total length of a network-qualified LU name cannot be greater than 17 characters in length (that is, the network ID and the network LU name, concatenated by a period, cannot exceed 17 characters).

System action

The system rejects the request.

Programmer response

Specify a correct LU name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60010I ATBTRACE SYNTAX ERROR: THE
TP KEYWORD VALUE IS MISSING
OR TOO LONG**

Explanation

On an ATBTRACE request, the TP keyword value was either missing or too long. The transaction program name cannot be greater than 64 characters.

System action

The system rejects the request.

Programmer response

Specify a correct TP name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60011I ATBTRACE SYNTAX ERROR: THE
SYMDEST KEYWORD VALUE IS
MISSING OR TOO LONG**

Explanation

On an ATBTRACE request, the SYMDEST keyword value was either missing or too long. The symbolic destination name cannot be greater than 8 characters.

System action

The system rejects the request.

Programmer response

Specify a correct symbolic destination name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60012I ATBTRACE SYNTAX ERROR: THE
USERID KEYWORD VALUE IS
MISSING OR TOO LONG**

Explanation

On an ATBTRACE request, the USERID keyword value was missing or too long. The user ID cannot be greater than 10 characters.

System action

The system rejects the request.

Programmer response

Specify a correct user ID on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60013I ATBTRACE SYNTAX ERROR: THE
DATASET KEYWORD VALUE IS
MISSING OR TOO LONG**

Explanation

On an ATBTRACE request, the DATASET keyword value was either missing or too long. The data set name cannot be greater than a total of 44 characters.

System action

The system rejects the request.

Programmer response

Specify a correct data set name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60014I ATBTRACE SYNTAX ERROR:
CHARACTERS IN TP KEYWORD
VALUE "tp_name" ARE NOT VALID**

Explanation

On an ATBTRACE request, the TP keyword value contains one or more characters that are not from character set 00640.

In the message text:

tp_name

The incorrect TP name specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

Correct the TP name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBVSTM

**ATB60015I ATBTRACE SYNTAX ERROR: THE
LU KEYWORD VALUE "lu_name" IS
NOT VALID**

Explanation

On an ATBTRACE request, the LU keyword value either:

- Contains one or more characters that are not from character set 00640 or the Type A character set, or
- Consists of a network ID and a network LU name, one of which is greater than 8 characters or contains one or more characters that are not from character set 00640.

In the message text:

lu_name

The incorrect LU name specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

Correct the LU name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBVSTM, ATBTRACE exec

**ATB60016I ATBTRACE SYNTAX ERROR:
CHARACTERS IN USERID
KEYWORD VALUE "userid" ARE
NOT VALID**

Explanation

On an ATBTRACE request, the USERID keyword value contains one or more characters that are not from the Type A character set.

In the message text:

userid

The incorrect user ID specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

Correct the user ID on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBVSTM

**ATB60017I ATBTRACE SYNTAX ERROR:
THE DATASET KEYWORD VALUE
"dsname" IS NOT VALID**

Explanation

On an ATBTRACE request, the DATASET keyword value does not meet all the syntax requirements for the data set name. If necessary, refer to [z/OS TSO/E User's Guide](#) for data set naming rules and conventions.

In the message text:

dsname

The incorrect data set name specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

Correct the data set name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBVSTM

**ATB60018I ATBTRACE SYNTAX
ERROR: CHARACTERS IN
SYMDEST KEYWORD VALUE
"sym_dest_name" ARE NOT VALID**

Explanation

On an ATBTRACE request, the SYMDEST keyword value contains one or more characters that are not from the Type A character set.

In the message text:

sym_dest_name

The incorrect symbolic destination name specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

Correct the symbolic destination name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBVSTM

**ATB60019I ATBTRACE SYNTAX ERROR:
MISMATCHED QUOTES IN
DATASET KEYWORD VALUE
"dsname"**

Explanation

On an ATBTRACE request, the DATASET keyword value was not enclosed in single quotes. A fully qualified data set name must begin and end with a single quote.

In the message text:

dsname

The keyword value that was found with mismatched quotes.

System action

The system rejects the request.

Programmer response

Correct the data set name on the ATBTRACE request, and re-issue it.

Detecting Module:

ATBTRACE exec

**ATB60020I ATBTRACE REQUEST
UNSUCCESSFUL: APPC/MVS
TRACE ROUTINE NOT FOUND**

Explanation

The system could not locate the APPC/MVS trace routine that processes ATBTRACE requests.

System action

The system rejects the request.

System programmer response

Make sure that the application is running on an OS/390 V1R3 (or later) system. If so, make sure that the APPC/MVS load module ATBVSTSS is in SYS1.LPALIB.

Detecting Module:

ATBTRACE exec

Programmer response

Contact your system programmer to diagnose the problem.

ATB60021I	ATBTRACE REQUEST UNSUCCESSFUL: INTERNAL SYSTEM ERROR IN ATBTRACE PROCESSING, RETURN CODE <i>return_code</i>
------------------	--

Explanation

The system encountered an internal error while processing an ATBTRACE request. Depending on the ATBTRACE processing that was in progress when the error occurred, the trace data set might contain trace data; if so, that data might be unusable.

In the message text:

return code

The return code for the internal error

System action

The system terminates ATBTRACE processing.

System programmer response

Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center, and provide the return code and the SVC dump, if one is available.

Detecting Module:

ATBTRACE exec

Programmer response

Contact your system programmer to diagnose the problem.

ATB60022I	ATBTRACE REQUEST UNSUCCESSFUL: APPC/MVS TRACE ROUTINE IS NOT AVAILABLE BECAUSE OF PREVIOUS INTERNAL ERROR
------------------	--

Explanation

An ATBTRACE request was invoked on a system on which the APPC/MVS trace routine is no longer active, because it previously encountered a non-retryable internal error.

System action

The system rejects the request.

Operator response

At the request of the system programmer, bring down and restart the APPC address space.

System programmer response

If you absolutely must have application program interface (API) tracing capability on this system, ask the operator to bring down and restart the APPC address space.

Detecting Module:

ATBVSTS

Programmer response

Contact your system programmer to determine whether to start tracing activity on another system, or restart tracing activity on this system.

ATB60023I	ATBTRACE START UNSUCCESSFUL: DYNAMIC ALLOCATION OF DATA SET <i>dsname</i> FAILED
------------------	---

Explanation

The system could not dynamically allocate the trace data set. A message with prefix IEC, IGD, or IKJ further explains the dynamic allocation error.

In the message text:

dsname

The data set name specified on the ATBTRACE request

System action

The system rejects the request.

Programmer response

To correct the problem, follow the instructions for the accompanying IEC, IGD, or IKJ message.

Detecting Module:

ATBVSTW

**ATB60024I ATBTRACE STOP REQUEST IS
QUEUED FOR COMPLETION**

Explanation

The system could not complete the stop request because of large number of API trace record yet to be written or because of I/O contention. You will not be able to view the trace data set until the trace stop request is completed by the system.

System action

The system partially completes the request.

Programmer response

Issue an ATBTRACE LIST request specifying the data set name you specified on the STOP request. If the STOP request is completed by the system, ATB60034I message will be returned. You can view the trace data set now. If the STOP request is not complete, system will return ATB60047I message.

Detecting Module:

ATBVSTE

**ATB60026I ATBTRACE NOT AVAILABLE: THE
APPC ADDRESS SPACE IS NOT
ACTIVE**

Explanation

An ATBTRACE request was invoked on a system on which the APPC address space is not active.

System action

The system rejects the request.

Operator response

At the request of the system programmer, restart APPC on this system.

System programmer response

Determine whether the APPC address space should be active on this system. If so, ask the operator to restart APPC.

Detecting Module:

ATBVSTS, ATBVSTT

Programmer response

Contact your system programmer to determine whether the APPC address space should be active on this system.

**ATB60027I ATBTRACE START
UNSUCCESSFUL: YOU ARE NOT
AUTHORIZED TO START THE
TRACE ASSOCIATED WITH LU
lu_name AND TP tp_name**

Explanation

The issuer of the ATBTRACE START request does not have authorization to start a trace for this LU and TP combination.

In the message text:

lu_name

The LU name specified on the ATBTRACE request

tp_name

The TP name specified on the ATBTRACE request

System action

The system rejects the request.

System programmer response

Issue the appropriate security product command to authorize the user to trace this LU and TP. See [z/OS MVS Planning: APPC/MVS Management](#) for more information on security requirements.

Detecting Module:

ATBVSTM

Programmer response

If you should have authority to trace this LU and TP, check with your security administrator. Once you have the proper authorization, re-issue the request.

**ATB60028I ATBTRACE START
UNSUCCESSFUL: THE ACTIVE
SIDE INFORMATION DATA SET
DOES NOT CONTAIN AN ENTRY
FOR sym_dest_name**

Explanation

The SYMDEST keyword value on an ATBTRACE request does not correspond to an entry in the active side information data set.

In the message text:

sym_dest_name

The symbolic destination name specified on the ATBTRACE request

System action

The system rejects the request.

System programmer response

Determine whether this symbolic destination name should be defined in the active side information data set.

Detecting Module:

ATBSD1G

Programmer response

If you specified a valid symbolic destination name as the SYMDEST keyword value, ask the system programmer to add the name to the active side information data set. Otherwise, correct the symbolic destination name, and re-issue the request.

ATB60029I	ATBTRACE START UNSUCCESSFUL: DYNAMIC ALLOCATION OF DATA SET <i>dsname</i> FAILED – RETURN CODE <i>retcode</i> ERROR REASON CODE <i>rsncode</i>
------------------	---

Explanation

The system could not dynamically allocate the trace data set, and could not issue a message that further explains the error.

In the message text:

dsname

The data set name specified on the ATBTRACE request

retcode

The return code from dynamic allocation.

rsncode

The error reason code from dynamic allocation.

System action

The system rejects the request.

Programmer response

To correct the problem, follow the instructions in [z/OS MVS Programming: Authorized Assembler Services Guide](#) for the dynamic allocation return and error reason code combination.

Detecting Module:

ATBVSTW

ATB60030I	ATBTRACE START UNSUCCESSFUL: <i>userid</i> DOES NOT HAVE UPDATE ACCESS TO DATA SET <i>dsname</i>
------------------	---

Explanation

The issuer of the ATBTRACE START request does not have update access to the data set specified on ATBTRACE request.

In the message text:

userid

The user ID under which the ATBTRACE START request was issued.

dsname

The data set name specified on the ATBTRACE START request

System action

The system rejects the request.

Programmer response

On the ATBTRACE request, specify the name of a data set to which you have update access, and re-issue the request.

Detecting Module:

ATBVSTW

ATB60031I	ATBTRACE STOP UNSUCCESSFUL: YOU ARE NOT AUTHORIZED TO STOP THE TRACE ASSOCIATED WITH LU <i>lu_name</i> AND TP <i>tp_name</i>
------------------	---

Explanation

The issuer of the ATBTRACE STOP request does not have authorization to stop a trace for this LU and TP combination.

In the message text:

lu_name

The name of the LU associated with this ATBTRACE request

tp_name

The name of the TP associated with this ATBTRACE request

System action

The system rejects the request.

System programmer response

Issue the appropriate security product command to authorize the user to trace this LU and TP. See [z/OS MVS Planning: APPC/MVS Management](#) for more information on security requirements.

Detecting Module:

ATBVSTM

Programmer response

Make sure you specified the correct data set name on the ATBTRACE request. If you should have authority to stop the trace associated with this LU and TP combination, check with your security administrator. (If the data set specified on the STOP request has active traces for additional LU/TP combinations, make sure you have authority to stop those LU/TP combinations as well, or the system might issue this message again after you re-issue the request.) Once you have the proper authorization, re-issue the request.

ATB60032I

**ATBTRACE START
UNSUCCESSFUL: AN ATBTRACE
STOP REQUEST IS PENDING FOR
DATA SET *dsname***

Explanation

ATBTRACE was invoked to start a trace, but the system is still processing an ATBTRACE STOP request that was already issued for the data set specified on the START request.

In the message text:

dsname

The data set specified on the ATBTRACE requests.

System action

The system rejects the request.

Programmer response

Issue an ATBTRACE LIST request, specifying the data set name you specified on the START request. When the system returns message ATB60034I, you can re-issue the START request.

Detecting Module:

ATBVSTT

ATB60033I

**ATBTRACE STOP UNSUCCESSFUL:
NO API TRACES ARE ACTIVE FOR
DATA SET *dsname***

Explanation

ATBTRACE was invoked to stop an application program interface (API) trace, but no trace is active for the specified data set, or the trace is already being stopped.

In the message text:

dsname

The data set specified on the ATBTRACE STOP request.

System action

The system rejects the request.

Programmer response

Determine whether the trace was already stopped, or was ever started, and take appropriate action.

Detecting Module:

ATBVSTM

ATB60034I

NO API TRACES ARE ACTIVE

Explanation

ATBTRACE was invoked to list any active application program interface (API) traces, but no traces are active on this system, or no traces are active for the data set specified on the ATBTRACE LIST request.

System action

The system processes the ATBTRACE request by returning this message; no list is produced because no traces are active.

Programmer response

Determine whether an ATBTRACE START was ever requested for the specified data set, or for any data set for this system, and take appropriate action.

Detecting Module:

ATBVSTL

**ATB60035I ATBTRACE START WAS
SUCCESSFUL**

Explanation

The system successfully processed an ATBTRACE START request.

System action

The system will collect trace data as instructed by the keyword values specified on the ATBTRACE request.

Programmer response

None.

Detecting Module:

ATBTRACE exec

**ATB60036I ATBTRACE STOP WAS
SUCCESSFUL**

Explanation

The system successfully processed an ATBTRACE STOP request.

System action

The system stops collecting any further trace data for the data set specified on the ATBTRACE request.

Programmer response

None.

Detecting Module:

ATBTRACE exec

**ATB60037I ATBTRACE REQUEST
UNSUCCESSFUL: APPC/MVS
PROCESSING ENCOUNTERED**

INTERNAL ERROR RETURN CODE

retcode REASON CODE rsncode

Explanation

APPC/MVS encountered an internal error while processing an ATBTRACE START, STOP, or LIST request. Depending on the ATBTRACE processing that was in progress when the error occurred, the trace data set might contain trace data; if so, that data might be unusable.

In the message text:

retcode

4-byte internal error return code.

rsncode

12-byte internal error reason code.

System action

The system terminates ATBTRACE processing.

System programmer response

Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center, and provide the return and reason codes and the SVC dump, if one is available.

Detecting Module:

**ATBVSTB, ATBVSTE, ATBVSTL,
ATBVSTM, ATBVSTT, ATBVSTW**

Programmer response

Contact your system programmer to diagnose the problem.

**ATB60038I ATBTRACE START
UNSUCCESSFUL: THE MAXIMUM
NUMBER OF API TRACE DATA
SETS IS ALREADY IN USE**

Explanation

When processing an ATBTRACE START request, the system determined that it was already collecting trace data in the maximum number of application program interface (API) trace data sets. The maximum is 50.

System action

The system rejects the request.

Programmer response

Issue ATBTRACE LIST without specifying any data set name, and examine the resulting list of active traces. If a STOP request is pending for a data set, you can issue the LIST request again, specifying the data set for which the STOP is in progress. When the system returns message ATB60034I, you can re-issue the START request.

If no STOP requests are in progress, examine the list for any active traces that aren't required any more. You can issue an ATBTRACE STOP for the data set associated with those traces, and re-issue the ATBTRACE START request. Keep in mind that issuing an ATBTRACE STOP stops **all** active traces to the data set specified on the STOP request.

Detecting Module:

ATBVSTT

**ATB60039I ATBTRACE REQUEST
UNSUCCESSFUL: THE REQUEST
WAS ISSUED WHILE IN SRB MODE**

Explanation

An ATBTRACE START, STOP, or LIST request was invoked from a program that is running in service request block (SRB) mode. An ATBTRACE request cannot be issued from an SRB.

System action

The system rejects the request.

Programmer response

Make sure any ATBTRACE request is issued from a program that is running in task mode. For example, if you want to leave the ATBTRACE request in the same program, you'll have to use a method other than the SCHEDULE or IEAMSCHD macro to run that program.

Detecting Module:

ATBVSTM

**ATB60040I ATBTRACE REQUEST
UNSUCCESSFUL: THE REQUEST
WAS ISSUED WHILE HOLDING
LOCKS**

Explanation

An ATBTRACE START, STOP, or LIST request was invoked from a program that was holding one or more locks. An ATBTRACE request cannot be issued from a program while it is holding any lock.

System action

The system rejects the request.

Programmer response

Make sure your program is not holding any locks when it issues an ATBTRACE request; then re-run the program.

Detecting Module:

ATBVSTM

**ATB60041I ATBTRACE REQUEST
UNSUCCESSFUL: THE REQUEST
WAS ISSUED WHILE IN CROSS
MEMORY MODE**

Explanation

An ATBTRACE START, STOP, or LIST request was invoked from a program that was in cross memory mode. An ATBTRACE request cannot be issued from a program while its home and primary address spaces are not the same.

System action

The system rejects the request.

Programmer response

Make sure your program's home and primary address spaces are the same when it issues an ATBTRACE request. (The secondary address space does not have to be the same as the home and primary address spaces.) Then re-run the program.

Detecting Module:

ATBVSTM

ATB60042I ATBTRACE LIST IS COMPLETE

Explanation

All the information about active traces has been listed for a specific ATBTRACE LIST request. This message appears at the end of the trace information.

System action

The system continues processing.

Programmer response

None.

Detecting Module:

ATBTRACE exec

**ATB60043I ATBTRACE START
UNSUCCESSFUL: TRACE DATA SET
dsname ALREADY IN USE BY user
FOR API TRACES**

Explanation

The issuer of an ATBTRACE START request specified the name of a data set that is already in use for application program interface (API) tracing by a different user. Only one user at a time can use the trace data set for an API trace.

In the message text:

dsname

The data set specified on the ATBTRACE START request.

user

The user ID under which the first ATBTRACE START request was issued for this data set. Additional ATBTRACE START requests for this data set can be issued only under this user ID.

System action

The system rejects the request.

Programmer response

Re-issue the ATBTRACE START with a different data set name or, if you have authority to do so, re-issue the ATBTRACE START from the user ID indicated by this message.

Detecting Module:

ATBVSTT

**ATB60044I ATBTRACE START
UNSUCCESSFUL: TRACE DATA SET
dsname IS NOT USABLE**

Explanation

On an ATBTRACE START request, the user specified a data set that the system cannot use.

In the message text:

dsname

The data set specified on the ATBTRACE START request.

System action

The system rejects the request, and issues a message with prefix AHL to further explain the error.

Programmer response

Follow the instructions for the AHL message to correct the error, and re-issue the ATBTRACE START request.

Detecting Module:

ATBVSTW

**ATB60045I ATBTRACE REQUEST
UNSUCCESSFUL: TRACE DATA SET
dsname IS NOT SEQUENTIAL**

Explanation

The issuer of an ATBTRACE START request specified the name of a partitioned data set (PDS or PDSE). The data set for application program interface (API) trace data must be sequential.

In the message text:

dsname

The data set specified on the ATBTRACE START request.

System action

The system rejects the request.

Programmer response

Re-issue the ATBTRACE START with a different data set name.

Detecting Module:

ATBVSTW, ATBTRACE exec

ATB60046I **LIST OF ACTIVE API TRACES FOR
DATA SET *dsname* IN USE BY
user API TRACE WAS STARTED
AT *mm/dd/yyyy hh:mm:ss.nnnnnn*
FOR:
LU: *lu_name*
TP: *tp_name*
SYMDEST: {*sym_dest_name*|N/A}
{USERID: {*userid*|*} | SECNONE}**

Explanation

For an ATBTRACE LIST request, the system displays active application program interface (API) traces. If a data set name was specified on the request, the system displays only those active traces associated with the data set. The text beginning with "API TRACE WAS STARTED..." and the subsequent information is repeated for each trace (that is, repeated as many times as the system wrote message ATB60051I to the data set for an ATBTRACE START request).

If a data set name was **not** specified, the system displays the entire message for each active API trace data set on the system, repeating the text beginning with "API TRACE WAS STARTED..." for each trace associated with each data set.

In the message text:

dsname

The name of a data set containing trace data.

user

The user ID under which the ATBTRACE START request was issued for this data set.

mm/dd/yyyy hh:mm:ss.nnnnnn

The month, date, year, hour, minute, second, and microsecond at which APPC/MVS processed the ATBTRACE START request for the API trace. The time is local time.

lu_name

The name of the logical unit (LU) associated with a currently active trace.

tp_name

The name of the transaction program (TP) associated with a currently active trace.

sym_dest_name

The symbolic destination name associated with a currently active trace. If the ATBTRACE START request specified a value for the SYMDEST keyword, that value is displayed for *sym_dest_name*, and the LU and TP values represented by *sym_dest_name* also appear for *lu_name* and *tp_name*, respectively. If the START

request did **not** specify the SYMDEST keyword and value (that is, the LU and TP keywords and values were specified), "N/A" appears to indicate that *sym_dest_name* does not apply.

userid

The user ID associated with a currently active trace. If the ATBTRACE START request did not specify a USERID keyword and value, an asterisk (*) appears for *userid*; if the START request specified the SECNONE keyword instead of the USERID keyword and value, SECNONE appears instead of USERID.

System action

The system continues processing.

Programmer response

None.

Detecting Module:

ATBVSTM

**ATB60047I ATBTRACE STOP IS IN PROGRESS
FOR DATA SET *dsname***

Explanation

An ATBTRACE LIST request was issued when APPC/MVS is still processing an ATBTRACE STOP request. In this case, the system does not include any application program interface (API) trace information for this data set in the output resulting from the ATBTRACE LIST request. If the ATBTRACE LIST request did not specify a data set name, this message is repeated for each data set for which an ATBTRACE STOP request is still in progress.

In the message text:

dsname

The name of a data set containing trace data.

System action

The system continues processing.

Programmer response

None.

Detecting Module:

ATBVSTM

ATB60048I **ATBTRACE STOP WAS
SUCCESSFUL, BUT AN ERROR
ENCOUNTERED WHILE CLOSING
THE DATA SET MIGHT HAVE
CAUSED TRACE DATA TO BE LOST**

Explanation

The system successfully processed an ATBTRACE STOP request; however, the system encountered an error while closing the data set. Because of the error, trace data might have been lost.

System action

The system stops collecting any further trace data for the data set specified on the ATBTRACE request.

Programmer response

None.

Detecting Module:

ATBVSTW

ATB60049I **API TRACE WAS STARTED AT
mm/dd/yyyy hh:mm:ss.nnnnnn
FOR:
LU: *lu_name*
TP: *tp_name*
SYMDEST: {*sym_dest_name*|N/A}
SECNONE**

Explanation

For ATBTRACE START requests for a particular data set, the system writes this message to the data set for each unique combination of LU and TP keyword values, with the SECNONE keyword. (If you specify the SYMDEST keyword instead of LU and TP, the system resolves the SYMDEST keyword value to actual LU and TP values before determining whether the START request specifies a unique LU/TP combination.)

If you issue a START request that is identical to a previous START request, without an intervening STOP request for this data set, the system returns message ATB60035I to indicate that the request was successful—tracing has already begun for that LU/TP combination.

In the message text:

mm/dd/yyyy hh:mm:ss.nnnnnn

The month, date, year, hour, minute, second, and microsecond at which APPC/MVS processed the

START request for the API trace. The time is local time.

lu_name

The name of the logical unit (LU) associated with a currently active trace.

tp_name

The name of the transaction program (TP) associated with a currently active trace.

sym_dest_name

The symbolic destination name associated with a currently active trace. If the ATBTRACE START request specified a value for the SYMDEST keyword, that value is displayed for *sym_dest_name*, and the LU and TP values represented by *sym_dest_name* also appear for *lu_name* and *tp_name*, respectively. If the START request did **not** specify the SYMDEST keyword and value (that is, the LU and TP keywords and values were specified), "N/A" appears to indicate that *sym_dest_name* does not apply.

System action

The system continues processing.

Programmer response

None.

Detecting Module:

ATBVSTT

ATB60050I **API TRACES STOPPED BY
APPC/MVS AT mm/dd/yyyy
hh:mm:ss.nnnnnn BECAUSE OF AN
APPC/MVS INTERNAL ERROR. THE
DATA SET CONTAINS TRACE DATA
FOR: API TRACE WAS STARTED
AT mm/dd/yyyy hh:mm:ss.nnnnnn
FOR:
LU: *lu_name*
TP: *tp_name*
SYMDEST: {*sym_dest_name*|N/A}
{USERID: {*userid*|*} | SECNONE }**

Explanation

If APPC/MVS encounters an internal error and stops the APPC/MVS trace routine, the system writes this message to any data set with active application program interface (API) traces. Any API trace entries that were collected but not written to a data set are lost.

In each data set, this message appears at the end of the most current trace data entries. (If the system had to wrap the trace entries, this message might not be the last message in the data set; a fragment of or entire trace entries for previous trace data might follow this message.

In each data set, the text beginning with "API TRACE STARTED..." and the subsequent information is repeated for each trace (that is, repeated as many times as the system wrote message ATB60051I to the data set for an ATBTRACE START request).

In the message text:

mm/dd/yyyy hh:mm:ss.nnnnnn

The month, date, year, hour, minute, second, and microsecond at which APPC/MVS processed the ATBTRACE START request for the API trace. The time is local time.

lu_name

The name of the logical unit (LU) associated with a currently active trace.

tp_name

The name of the transaction program (TP) associated with a currently active trace.

sym_dest_name

The symbolic destination name associated with a currently active trace. If the ATBTRACE START request specified a value for the SYMDEST keyword, that value is displayed for *sym_dest_name*, and the LU and TP values represented by *sym_dest_name* also appear for *lu_name* and *tp_name*, respectively. If the START request did **not** specify the SYMDEST keyword and value (that is, the LU and TP keywords and values were specified), "N/A" appears to indicate that *sym_dest_name* does not apply.

userid

The user ID associated with a currently active trace. If the ATBTRACE START request did not specify a USERID keyword and value, an asterisk (*) appears for *userid*; if the START request specified the SECNONE keyword instead of the USERID keyword and value, SECNONE appears instead of USERID.

System action

The system stops all active API traces because of APPC/MVS trace routine termination, and issues message ATB499I to the console.

Programmer response

None.

Detecting Module:

ATBVSTW

ATB60051I

API TRACE WAS STARTED AT

mm/dd/yyyy hh:mm:ss.nnnnnn

FOR:

LU: *lu_name*

TP: *tp_name*

SYMDEST: {*sym_dest_name*|N/A}

USERID: {*userid*|*}

Explanation

For ATBTRACE START requests for a particular data set, the system writes this message to the data set for each unique combination of LU, TP, and USERID keyword values. (If you specify the SYMDEST keyword instead of LU and TP, the system resolves the SYMDEST keyword value to actual LU and TP values before determining whether the START request specifies a unique LU/TP/USERID combination.)

If you issue a START request that is identical to a previous START request, without an intervening STOP request for this data set, the system returns message ATB60035I to indicate that the request was successful — tracing has already begun for that LU/TP/USERID combination.

In the message text:

mm/dd/yyyy hh:mm:ss.nnnnnn

The month, date, year, hour, minute, second, and microsecond at which APPC/MVS processed the START request for the API trace. The time is local time.

lu_name

The name of the logical unit (LU) associated with a currently active trace.

tp_name

The name of the transaction program (TP) associated with a currently active trace.

sym_dest_name

The symbolic destination name associated with a currently active trace. If the ATBTRACE START request specified a value for the SYMDEST keyword, that value is displayed for *sym_dest_name*, and the LU and TP values represented by *sym_dest_name* also appear for *lu_name* and *tp_name*, respectively. If the START request did **not** specify the SYMDEST keyword and value (that is, the LU and TP keywords and values were specified), "N/A" appears to indicate that *sym_dest_name* does not apply.

userid

The user ID associated with a currently active trace. If the ATBTRACE START request did not specify a USERID keyword and value, an asterisk (*) appears for *userid*.

System action

The system continues processing.

Programmer response

None.

Detecting Module:

ATBVSTT

ATB60052I **ATBTRACE STOP REQUEST
ISSUED BY user AT mm/dd/yyyy
hh:mm:ss.nnnnnnn. THE DATA SET
CONTAINS TRACE DATA FOR: API
TRACE WAS STARTED AT mm/dd/
yyyy hh:mm:ss.nnnnnnn FOR:
LU: lu_name
TP: tp_name
SYMDEST: {sym_dest_name|N/A}
{USERID: {userid|*} | SECNONE }**

Explanation

For an ATBTRACE STOP request, the system writes this message to the data set specified on the STOP request. This message appears at the end of the most current application program interface (API) trace data entries. (If the system had to wrap the trace entries, this message might not be the last message in the data set; a fragment of or entire trace entries for previous trace data might follow this message.)

The text beginning with "API TRACE STARTED..." and the subsequent information is repeated for each trace (that is, repeated as many times as the system wrote message ATB60051I to the data set for an ATBTRACE START request).

In the message text:

user

The user ID of the issuer of the ATBTRACE STOP request for this data set.

mm/dd/yyyy hh:mm:ss.nnnnnnn

The month, date, year, hour, minute, second, and microsecond at which APPC/MVS processed the ATBTRACE START request for the API trace. The time is local time.

lu_name

The name of the logical unit (LU) associated with a currently active trace.

tp_name

The name of the transaction program (TP) associated with a currently active trace.

sym_dest_name

The symbolic destination name associated with a currently active trace. If the ATBTRACE START request specified a value for the SYMDEST keyword, that value is displayed for *sym_dest_name*, and the LU and TP values represented by *sym_dest_name* also appear for *lu_name* and *tp_name*, respectively. If the START request did **not** specify the SYMDEST keyword and value (that is, the LU and TP keywords and values were specified), "N/A" appears to indicate that *sym_dest_name* does not apply.

userid

The user ID associated with a currently active trace. If the ATBTRACE START request did not specify a USERID keyword and value, an asterisk (*) appears for *userid*; if the START request specified the SECNONE keyword instead of the USERID keyword and value, SECNONE appears instead of USERID.

System action

The system stops all active API traces associated with this data set.

Programmer response

None.

Detecting Module:

ATBVSTW

ATB60054I **API TRACE WAS SUSPENDED
AT mm/dd/yyyy hh:mm:ss.nnnnnnn,
number ENTRIES WERE LOST**

Explanation

When the system becomes overloaded with entries for a particular application program interface (API) trace, it suspends the trace until it can write the entries it has already collected to the data set. The system issues this message only if trace entries were lost while the trace was suspended.

The system might suspend an API trace because conversation activity for the TPs being traced is very high, because the data set block size is not optimal

for the volume of entries, or because of environmental constraints. For example, if conversation activity is high and many trace entries are generated, but I/O to the data set has stopped because another system issued a RESERVE for the volume on which that data set resides, the system might temporarily suspend the trace. Similarly, system tuning problems might also cause the system to suspend tracing activity.

In the message text:

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS suspended the API traces. The time is local time.

number

The number of trace entries that were lost. The maximum value for this number is 2147483648; once the system reaches the maximum value, it resets the counter to zero and continues to count lost entries.

System action

When it issues this message, the system has been able to resume the trace already. It continues collecting and writing current trace data to the data set.

Programmer response

If you determine that you have lost important trace entries, you can stop the trace and restart it after making some adjustments to avoid losing entries. For example, you might issue an ATBTRACE LIST request and examine the resulting information, only to find you are tracing more conversations than you need. In that case, you might reduce tracing activity by specifying a different value for the USERID keyword on the START request.

Alternatively, you could allocate another data set without specifying the block size, and issue a START request with the name of that data set. Then the system will select a block size that is optimal for the tracing activity.

Detecting Module:

ATBVSTW

ATB60055I ENTRY TO THE *service_name* SERVICE:

Explanation

TIMESTAMP: *mm/dd/yyyy hh:mm:ss.nnnnnn*

ASID: *asid*

TCB ADDR: *tcb_address*

JOB NAME: *jobname*

LU: *lu_name*

TP: *tp_name*

USERID: *userid*

CONVID: *conversation_id*

PARAMETERS: *list_of_parameters*

In a conversation being traced, a transaction program (TP) issued a call to an APPC/MVS or CPI-C service identified by *service_name*. (Refer to [Table 10 on page 81](#) for list of supported services.) This message contains the following details about the call on entry to that service:

service_name

The call name of the APPC/MVS callable service or CPI-C verb that was issued.

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS received the service call. The time is local time.

asid

The address space identifier (ASID) associated with the TP that issued the call.

tcb_address

The address of the task that called the service. This value is zero if a service request block (SRB) routine called the service.

jobname

The job name associated with the TP that issued the call.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

The user ID associated with the address space where the TP is running.

conversation_id

The hexadecimal value that the system uses to identify a particular conversation. If zero appears in this field, either service call does not contain a parameter for the conversation ID, or the caller specified zero as the conversation ID.

list_of_parameters

The **supplied** parameters that were specified on the service call. Refer to individual service descriptions in the appropriate APPC/MVS or CPI-C reference document for descriptions of the

parameters for which the caller must supply a value on the service call.

System action

The system continues processing.

Programmer response

None.

Detecting Module:	
ATBVSTW	

ATB60056I	THE <i>service_name</i> SERVICE COMPLETED.
------------------	---

Explanation

TIMESTAMP: <i>mm/dd/yyyy hh:mm:ss.nnnnnnn</i>
ASID: <i>asid</i>
TCB ADDR: <i>tcb_address</i>
JOB NAME: <i>jobname</i>
LU: <i>lu_name</i>
TP: <i>tp_name</i>
USERID: <i>userid</i>
CONVID: <i>conversation_id</i>
PARAMETERS: <i>list_of_parameters</i>
[ERROR_INFO:
MESSAGE_TEXT_LENGTH: <i>msg_text_length</i>
MESSAGE_TEXT: <i>msg_text</i>
[ERROR_LOG_PRODUCT_SET_ID_LENGTH:
<i>id_length</i>
ERROR_LOG_PRODUCT_SET_ID: <i>product_id</i>
ERROR_LOG_INFORMATION_LENGTH:
<i>info_length</i>
ERROR_LOG_INFORMATION: <i>log_info</i>]

In a conversation being traced, a transaction program (TP) issued a call to an APPC/MVS or CPI-C service identified by *service_name*. (Refer to [Table 10 on page 81](#) for list of supported services.) This message contains the following details about the call when all processing for that service has completed:

service_name

The call name of the APPC/MVS callable service or CPI-C verb that was issued.

mm/dd/yyyy hh:mm:ss.nnnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS completed the service call. The time is local time.

asid

The address space identifier (ASID) associated with the TP that issued the call.

tcb_address

The address of the task that called the service. This value is zero if a service request block (SRB) routine called the service.

jobname

The job name associated with the TP that issued the call.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

The user ID associated with the address space where the TP is running.

conversation_id

The hexadecimal value that the system uses to identify a particular conversation. If zero appears in this field, either service call does not contain a parameter for the conversation ID, or the caller specified zero as the conversation ID.

list_of_parameters

The parameters for which the system returns specific values after processing the service call. If an error is encountered for the service call, the only parameters included in the list are the return code and reason code, if a reason code applies. Refer to individual service descriptions in the appropriate APPC/MVS or CPI-C reference document for descriptions of the parameters for which the system returns specific values.

msg_text_length

The total number of characters that appear in *msg_text*.

msg_text

A message that describes an error on the call to the service identified by *service_name*. The data returned for this parameter will appear as a message in the format ATB8xxxxI. See “[Error Extract \(ATB8\) Messages](#)” on page 343 for explanations of messages returned by APPC/MVS.

id_length

The length of the value *product_id*.

- If no product set ID information is available from the partner system, APPC/MVS sets the value on this parameter to zero.
- If product set ID information is available from the partner system, APPC/MVS sets the value on this parameter to a number from 1 through 256.

If more than 256 bytes of product set ID information is available, APPC/MVS returns only the first 256 bytes of that information.

product_id

The identifier of the partner system that supplies the error log information. If the product set ID is more than 256 bytes long, APPC/MVS returns only the first 256 bytes of the product set ID.

For information about the format of a product set ID, see the descriptions of the Product Set ID (X'10') and the Product Identifier (X'11') MS Common Subvectors in *SNA Formats*

info_length

The length of the log information received from a partner TP or system. If no error log information is available from the partner TP or system, APPC/MVS sets the value on this parameter to zero.

log_info

A message that describes an error found by a partner system or TP. If APPC/MVS is the partner system that supplies this error log information, the data returned for this parameter will appear as a message in the format ASB7xxxxI or ATB7xxxxI. See “Error_Extract Error Log Information (ASB, ATB7) Messages” on page 317 for explanations of messages returned by APPC/MVS.

System action

The system continues processing.

Programmer response

None.

Detecting Module:
ATBVSTW

**ATB60057I SYNCHRONOUS RETURN FROM
THE service_name SERVICE.**

Explanation

TIMESTAMP: mm/dd/yyyy hh:mm:ss.nnnnnn
ASID: asid

TCB ADDR: tcb_address

JOB NAME: jobname

LU: lu_name

TP: tp_name

USERID: userid

CONVID: conversation_id

In a conversation being traced, a transaction program (TP) issued a call to an APPC/MVS service identified by *service_name*, specifying a Notify_type of Notify_ECB. (Refer to [Table 10 on page 81](#) for list of supported services.) In this case, the system returns control to the TP before processing the call. This message indicates that the system has accepted the call for asynchronous processing, and includes the following details about the call on **initial** return from the system:

service_name

The call name of the APPC/MVS callable service or CPI-C verb that was issued.

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS first returns control to the program that issued the service call. The time is local time.

asid

The address space identifier (ASID) associated with the TP that issued the call.

tcb_address

This value is zero.

jobname

The job name associated with the TP that issued the call.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

The user ID associated with the address space where the TP is running.

conversation_id

The hexadecimal value that the system uses to identify a particular conversation. If zero appears in this field, either service call does not contain a parameter for the conversation ID, or the caller specified zero as the conversation ID.

System action

The system continues processing. When the system finishes processing the call, it writes message ATB60056I to the data set.

Programmer response

None.

Detecting Module:
ATBVSTW

ATB60058I **ATBTRACE STOP WAS SUCCESSFUL, BUT *number* ENTRIES WERE LOST AT *mm/dd/yyyy hh:mm:ss.nnnnnn* BECAUSE OF BUFFER SHORTAGE**

Explanation

The system successfully processed an ATBTRACE STOP request; however, the system was unable to write all trace entries to the data set because of a buffer shortage.

In the message text:

number

The number of trace entries that were lost. The maximum value for this number is 2147483648 entries.

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS lost the API trace data entries. The time is local time.

System action

The system stops collecting any further trace data for the data set specified on the ATBTRACE request.

Programmer response

None.

Detecting Module:
ATBVSTW

ATB60061I **AN FMH-5 WAS SENT TO PARTNER LU *partner_lu*.**

Explanation

TIMESTAMP: *mm/dd/yyyy hh:mm:ss.nnnnnn*

ASID: *asid*

TCB ADDR: *tcb_address*

JOB NAME: *jobname*

LU: *lu_name*

TP: *tp_name*

USERID: *userid*

CONVID: *conversation_id*

FMH-5: *fmh-5*

For a conversation being traced, APPC/MVS sent an FMH-5 to the partner LU.

In the message text:

partner_lu

The 17-character name of the partner LU to which the FMH-5 was sent.

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS sent the FMH-5. The time is local time.

asid

The address space identifier (ASID) of the APPC address space.

tcb_address

The address of the task that sent the FMH-5.

jobname

The job name associated with the task that sent the FMH-5.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

The user ID associated with the address space where the TP is running.

conversation_id

The hexadecimal value that the system uses to identify a particular conversation.

fmh-5

256 bytes of hexadecimal FMH-5 data. Refer to *SNA Formats* for more information on FMH-5 format and contents.

System action

The system continues processing.

Programmer response

None.

Detecting Module:
ATBVSTW

ATB60062I **AN FMH-5 WAS RECEIVED ON
LOCAL LU *local_lu* FROM PARTNER
LU *partner_lu*.**

Explanation

TIMESTAMP: <i>mm/dd/yyyy hh:mm:ss.nnnnnnn</i>
ASID: <i>asid</i>
TCB ADDR: <i>tcb_address</i>
JOB NAME: <i>jobname</i>
LU: <i>lu_name</i>
TP: <i>tp_name</i>
USERID: <i>userid</i>
CONVID: <i>conversation_id</i>
FMH-5: <i>fmh-5</i>

For a conversation being traced, APPC/MVS received an FMH-5 from the partner LU.

In the message text:

partner_lu

The 17-character name of the partner LU that sent the FMH-5.

mm/dd/yyyy hh:mm:ss.nnnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS received the FMH-5. The time is local time.

asid

The address space identifier (ASID) of the APPC address space.

tcb_address

The address of the task that received the FMH-5.

jobname

The job name associated with the task that received the FMH-5.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

UserId associated with the APPC/MVS address space.

conversation_id

This value is zero.

fmh-5

256 bytes of hexadecimal FMH-5 data. Refer to *SNA Formats* for more information on FMH-5 format and contents.

System action

The system continues processing.

Programmer response

None.

Detecting Module:
ATBVSTW

ATB60063I **AN FMH-7 WAS SENT TO PARTNER
LU *partner_lu*.**

Explanation

TIMESTAMP: <i>mm/dd/yyyy hh:mm:ss.nnnnnnn</i>
ASID: <i>asid</i>
TCB ADDR: <i>tcb_address</i>
JOB NAME: <i>jobname</i>
LU: <i>lu_name</i>
TP: <i>tp_name</i>
USERID: <i>userid</i>
CONVID: <i>conversation_id</i>
FMH-7: <i>fmh-7</i>
ERROR_INFO:
MESSAGE_TEXT_LENGTH: <i>msg_text_length</i>
MESSAGE_TEXT: <i>msg_text</i>

For a conversation being traced, APPC/MVS sent an FMH-7 to the partner LU.

In the message text:

partner_lu

The 17-character name of the partner LU to which the FMH-7 was sent.

mm/dd/yyyy hh:mm:ss.nnnnnn

Is the month, date, year, hour, minute, second, and microsecond at which APPC/MVS sent the FMH-7. The time is local time.

asid

The address space identifier (ASID) of the APPC address space.

tcb_address

The address of the task that sent the FMH-7.

jobname

The job name associated with the task that sent the FMH-7.

lu_name

The name of the logical unit (LU) associated with the currently active trace.

tp_name

The name of the TP associated with the currently active trace.

userid

Userid associated with the APPC/MVS address space.

conversation_id

This value is zero.

fmh-7

The 7-character FMH-7 data, which includes the sense code back to VTAM. Refer to *SNA*

Formats for more information on FMH-7 format and contents.

msg_text_length

The total number of characters that appear in *msg_text*.

msg_text

The explanation of an error that occurred during the processing of the service call, in the form of an ATB7xxxxI message. For detailed information about ATB messages, refer to [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77.](#)

System action

The system continues processing.

Programmer response

None.

Detecting Module:**ATBVSTW**

Chapter 11. Error_Extract Reason Codes and Messages

This chapter complements the Error_Extract information presented in [Chapter 6, “Diagnosing Problems with APPC/MVS TPs,”](#) on page 77. It contains:

- A table that matches Error_Extract reason codes with Error_Extract error messages
- Explanations of Error_Extract error log information (ASB, ATB7) messages
- Explanations of Error_Extract (ATB8) messages.

Summary of Error_Extract Reason Codes

When the Error_Extract service finds an error in a call to a conversation service, it returns a reason code on the Service_reason_code parameter and an error message on the Message_text parameter.

Use the following table to find the reason codes that are associated with each message. Then see [“Error_Extract \(ATB8\) Messages”](#) on page 343 for the explanation of the message. Design your application to handle the error situation indicated in the message explanation.

There are no reason codes associated with the log data messages that Error_Extract returns.

Reason Code (decimal)	Associated Message Identifier
3	ATB80003I
4	ATB80004I
5	ATB80005I
6	ATB80006I
8	ATB80008I
9	ATB80009I
10	ATB80010I
11	ATB80011I
12	ATB80016I
13	ATB80016I
14	ATB80014I
15	ATB80011I
16	ATB80016I
17	ATB80016I
18	ATB80016I
19	ATB80016I
20	ATB80020I
21	ATB80016I
22	ATB80016I
23	ATB80023I

Reason Code (decimal)	Associated Message Identifier
24	ATB80016I
25	ATB80025I
26	ATB80026I
27	ATB80014I
28	ATB80016I
29	ATB80016I
30	ATB80023I
31	ATB80023I
33	ATB80033I
34	ATB80034I
35	ATB80016I
36	ATB80036I
37	ATB80037I
38	ATB80038I
39	ATB80039I
40	ATB80040I
41	ATB80041I
42	ATB80042I
43	ATB80043I
44	ATB80044I
45	ATB80045I
46	ATB80046I
47	ATB80047I
48	ATB80048I
49	ATB80049I
50	ATB80050I
51	ATB80051I
52	ATB80052I
53	ATB80053I
54	ATB80054I
55	ATB80055I
56	ATB80055I
58	ATB80058I
59	ATB80059I
60	ATB80060I

Reason Code (decimal)	Associated Message Identifier
61	ATB80061I
62	ATB80062I
63	ATB80063I
64	ATB80064I
65	ATB80065I
66	ATB80066I
67	ATB80067I
68	ATB80068I
69	ATB80069I
70	ATB80070I
71	ATB80071I
72	ATB80016I
73	ATB80073I
74	ATB80074I
75	ATB80075I
76	ATB80076I
77	ATB80077I
78	ATB80078I
79	ATB80079I
82	ATB80082I
83	ATB80083I
84	ATB80084I
85	ATB80085I
86	ATB80086I
87	ATB80087I
88	ATB80088I
89	ATB80089I
90	ATB80090I
91	ATB80091I
92	ATB80092I
93	ATB80093I
94	ATB80094I
95	ATB80095I
96	ATB80096I
97	ATB80097I

Reason Code (decimal)	Associated Message Identifier
98	ATB80098I
99	ATB80099I
100	ATB80100I
101	ATB80101I
102	ATB80102I
103	ATB80103I
104	ATB80104I
105	ATB80105I
106	ATB80106I
107	ATB80107I
108	ATB80108I
109	ATB80109I
110	ATB80110I
111	ATB80111I
112	ATB80112I
114	ATB80114I
115	ATB80115I
116	ATB80116I
117	ATB80117I
118	ATB80016I
119	ATB80119I
121	ATB80121I
122	ATB80122I
123	ATB80123I
124	ATB80124I
125	ATB80125I
126	ATB80126I
127	ATB80127I
127	ATB80127I
128	ATB80128I
129	ATB80129I
130	ATB80130I
131	ATB80131I
133	ATB80133I
134	ATB80134I

Reason Code (decimal)	Associated Message Identifier
135	ATB80135I
136	ATB80136I
138	ATB80138I
139	ATB80139I
140	ATB80140I
141	ATB80141I
142	ATB80142I
143	ATB80143I
144	ATB80144I

Error_Extract Error Log Information (ASB, ATB7) Messages

The Error_Extract service can return one of the messages described in this section when a partner system or TP finds an error in the most recently completed call to another APPC TP conversation service or CPI Communications (CPI-C) call. These messages appear on the Error_log_information parameter on return from Error_Extract.

Error_Extract also returns error messages when APPC/MVS finds an error in a call to an APPC conversation service. For descriptions of those messages, see [“Error_Extract \(ATB8\) Messages” on page 343](#).

ASB70000I **Abend occurred in APPC/MVS transaction scheduler while processing inbound allocate request.**

Explanation

A TP called the Allocate service to allocate a conversation with a program on MVS. An abend occurred in the APPC/MVS transaction scheduler when it tried to schedule the request.

System action

The system ends the conversation. The system rejects the request with a sense code of tp_not_available_no_retry (X'084C0000'). The system writes a logrec data set record that describes the error.

System programmer response

Check the logrec data set record for the abend that occurred in APPC/MVS transaction scheduler. See the response for the abend in [z/OS MVS System Codes](#).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ASBSCPR, ASBSCSS

ASB70001I **APPC/MVS transaction scheduler cannot obtain storage for its internal data area.**

Explanation

The APPC/MVS transaction scheduler could not obtain enough storage to process a request to schedule an inbound TP.

System action

The APPC/MVS scheduler ends the conversation. The system rejects the request with a sense code of tp_not_available_no_retry (X'084C0000'). The system writes a logrec data set record that describes the error.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module: ASBSCPR
--

ASB70002I **APPC/MVS cannot schedule inbound TP. Scheduler class *class_name* is not active.**

Explanation

The APPC/MVS transaction scheduler could not schedule an inbound TP using the specified scheduler class. The scheduler class is not active.

In the message text:

class_name

The name of the scheduler class specified in the TP profile for the inbound TP.

System action

The system ends the conversation. The system rejects the request with a sense code of `tp_not_available_no_retry (X'084C0000')`.

Operator response

At the request of the system programmer, enter a SET ASCH command to activate an updated ASCHPMxx parmlib member.

System programmer response

At the request of the application programmer, ensure that a CLASSADD statement in an ASCHPMxx parmlib member defines the scheduler class. See the ASCHPMxx parmlib member description in [z/OS MVS Initialization and Tuning Reference](#) for the syntax of the CLASSADD statement. Ask the operator to enter a SET ASCH command to activate the scheduler class.

Programmer response

If the TP is using a scheduler class that is specified in an associated TP profile, validate that the specified class is correct. If it is correct, or if the TP is using a default scheduler class, contact the system programmer.

Source

APPC/MVS

Detecting Module: ASBSCPR
--

ASB70003I **Scheduler class *class_name* is not defined to the APPC/MVS transaction scheduler.**

Explanation

The APPC/MVS transaction scheduler could not schedule an inbound TP using the scheduler class specified in the associated TP profile. The scheduler class is not defined in an ASCHPMxx parmlib member.

In the message text:

class_name

The name of the scheduler class specified in the TP profile for the inbound TP.

System action

The system ends the conversation. The system rejects the request to schedule the TP with a sense code of `tp_not_available_no_retry (X'084C0000')`.

System programmer response

Enter a CLASSADD statement in an ASCHPMxx parmlib member to define the scheduler class. See the ASCHPMxx parmlib member description in [z/OS MVS Initialization and Tuning Reference](#) for the syntax of the CLASSADD statement.

Programmer response

Verify that the scheduler class specified on the CLASS statement in the associated TP profile is correct. If it is correct, contact the system programmer.

Source

APPC/MVS

Detecting Module: ASBSCPR
--

ASB70004I **APPC/MVS transaction scheduler is terminating.**

Explanation

The APPC/MVS transaction scheduler could not schedule a TP. The scheduler is terminating.

System action

The system ends the conversation. The system rejects the request to schedule the TP with a sense code `tp_not_available_no_retry` (X'084C0000').

Operator response

Wait for the ASCH address space to end, as indicated by message ASB053I. Then, if you wish to restart the ASCH address space, enter a START ASCH command.

Programmer response

Contact the operator.

Source

APPC/MVS

Detecting Module:

ASBSCPR

ASB70005I **APPC/MVS transaction scheduler cannot schedule an inbound TP. Internal error code: *error_code***

Explanation

The APPC/MVS transaction scheduler could not schedule an inbound allocate request. An internal error occurred in the transaction scheduler.

In the message text:

error_code

An internal error code (decimal) that is useful to the IBM Support Center when diagnosing the error.

System action

The system ends the conversation. The system rejects the request with a sense code of `tp_not_available_no_retry` (X'084C0000').

System programmer response

Contact the IBM Support Center. Provide the error code identified in the message text.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCPR, ASBSCIS, ASBSCT2

ASB70006I **APPC/MVS transaction scheduler class not specified in TP Profile, and no default class name is defined.**

Explanation

The APPC/MVS transaction scheduler could not schedule an inbound TP. The associated TP profile did not specify a scheduler class, and no default scheduler class was defined on the OPTIONS statement in the ASCHPMxx parmlib member. Therefore the scheduler could not determine the class in which to schedule the TP.

System action

The system ends the conversation. The system rejects the request with a sense code of `tp_not_available_no_retry` (X'084C0000').

Operator response

At the request of the system programmer, enter a SET ASCH=xx command to activate the updated ASCHPMxx parmlib member.

System programmer response

Do one of the following:

- Enter a TPMODIFY command to add the name of a scheduler class to the CLASS statement in the TP profile
- Enter the OPTIONS DEFAULT(*class_name*) statement in an ASCHPMxx parmlib member, where *class_name* is the the name of the default scheduler class. Ask the operator to enter a SET ASCH=xx command to activate the updated ASCHPMxx parmlib member.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCPR

ASB70007I **APPC/MVS transaction scheduler cannot schedule inbound allocate request. Error codes**

from **IXCMSGI** macro: Return code *return_code*, reason code *reason_code*.

Explanation

A TP called the Allocate service to allocate a conversation with a program on MVS. The APPC/MVS transaction scheduler could not schedule the request. The system could not execute the IXCMSGI macro, which allows a cross-system coupling facility (XCF) member to receive a message from another member in its XCF group.

In the message text:

return_code

The return code from the IXCMSGI macro (in hexadecimal).

reason_code

The reason code from the IXCMSGI macro (in hexadecimal).

System action

APPC/MVS ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the Allocate service.

System programmer response

See the description of the IXCMSGI macro in *z/OS MVS Programming: Sysplex Services Reference* for the actions associated with the return and reason codes from IXCMSGI. If the problem persists after you take the specified actions, contact the IBM Support Center.

Programmer response

Contact the system programmer. Provide the return and reason codes from the IXCMSGI macro.

Source

APPC/MVS

Detecting Module:

ASBSCPR

ASB70008I **APPC/MVS transaction scheduler cannot schedule inbound allocate request. An internal error occurred.**

Explanation

A TP called the Allocate service to allocate a conversation with a program on MVS. The APPC/MVS transaction scheduler could not schedule the request because an internal error occurred.

System action

APPC/MVS ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Contact the IBM support center.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCPR

ASB70009I **APPC/MVS transaction scheduler cannot schedule inbound allocate request. Scheduler class *class_name* was deleted.**

Explanation

A TP called the Allocate service to allocate a conversation with a program on MVS. The APPC/MVS transaction scheduler could not schedule the program. The CLASS statement in the TP profile specified a scheduler class that was deleted before the TP received control. The scheduler cannot schedule the TP until the scheduler class is activated.

In the message text:

class_name

The name of the scheduler class specified in the TP profile for the inbound TP.

System action

APPC/MVS ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

Operator response

Enter a DISPLAY ASCH command to confirm that the scheduler class was deleted. If the scheduler

was incorrectly deleted, ask the system programmer to enter a CLASSADD statement in the ASCHPMxx parmlib member to add the class. Then enter a SET ASCH command to activate the class.

System programmer response

If requested by the operator, enter a CLASSADD statement in the ASCHPMxx parmlib member to add the class that was originally deleted.

Programmer response

Validate that the CLASS statement in the TP profile is correct. If it is correct, contact the operator to determine why the scheduler class was deleted. If it is not correct, enter a valid scheduler class name on the CLASS statement in the TP profile.

Source

APPC/MVS

Detecting Module:

ASBSCAD

ASB70010I **APPC/MVS transaction scheduler could not associate a TP with an initiator.**

Explanation

The APPC/MVS transaction scheduler could not schedule an inbound TP using the specified scheduler class. The scheduler could not associate the TP with an initiator.

System action

The system ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Contact the IBM Support Center.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCIS

ASB70011I **ASCH initiator could not create security environment for TP.**

Explanation

The APPC/MVS transaction scheduler could not create an execution environment for a TP. If the TP is a multi-trans TP, the TP profile might contain a generic user ID (specified on the GENERIC_ID statement) that is not valid.

System action

The system ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000'). The system might write a logrec data set record that describes the error. The system also might request an SVC dump.

System programmer response

Check the logrec data set to see if the system wrote an error record. If not, contact the support center for the security product that is installed on your system. Provide the SVC dump (if one is available).

Source

APPC/MVS

Detecting Module:

ASBSCIS

ASB70012I **An internal failure occurred in the APPC/MVS transaction scheduler. Detection code is: error_data**

Explanation

An internal failure occurred in APPC/MVS transaction scheduler (ASCH) processing.

In the message text:

error_data

An internal code (in hexadecimal), which is useful to the IBM Support Center when diagnosing the error.

System action

The system ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Contact the IBM Support Center. Provide the error code specified in the message text.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCIS

ASB70013I ASCH initiator was unable to determine the SYSOUT and accounting information for the TP.

Explanation

The APPC/MVS transaction scheduler could not obtain SYSOUT and accounting information for a user. The WORK ATTRIBUTES segment of the user's security profile does not contain the SYSOUT and accounting information required to schedule the TP.

System action

The system ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

At the request of the system administrator, contact the IBM Support Center.

Programmer response

Contact the system administrator.

System Administrator Response: At the request of the application programmer, specify valid SYSOUT and accounting information in the WORK ATTRIBUTES segment of the user's security profile. If the problem persists, contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCIS

ASB70014I ASCH initiator could not schedule a TP. TP account number rejected by IEFUAV installation exit.

Explanation

The APPC/MVS transaction scheduler (ASCH) initiator tried to create an execution environment for a TP. The initiator could not create the environment. The IEFUAV installation exit rejected the account number specified on the JCL JOB statement in one of the following:

- The associated TP profile
- The user's security profile.

System action

The system ends the conversation. The system rejects the request to schedule the TP with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

At the request of the application programmer, see [z/OS MVS Installation Exits](#) for a description of installation specifications for the IEFUAV exit. Ensure that the IEFUAV exit is working correctly.

Programmer response

Validate that the account number in the TP profile is correct. If it is correct, contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCIS

ASB70015I ASCH initiator cannot schedule the OpenEdition MVS fork TP. OpenEdition MVS job initiation processing return code is: error_data.

Explanation

An internal failure occurred in z/OS UNIX System Services initiation processing for the z/OS UNIX System Services fork TP.

error_data

An internal code (in hexadecimal), which is useful to the IBM Support Center when diagnosing the error.

System action

The system ends the conversation. The system rejects the request with a sense code of TP_not_available_no_retry (XX'084C0000').

System programmer response

Contact the IBM Support Center. Provide the error code *error_data* specified in the message text.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ASBSCIS

ATB70001I **TP name specified on inbound allocate request is not valid. Length of TP name is zero.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the TP name length contains a zero. The value in this field must be greater than zero.

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70002I **TP name specified on inbound allocate request is not valid. Name contains all blank characters.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the TP name contains all blank characters. The TP name must contain at least one character that is not a blank (if it is a SNA service TP) or no blank characters (if it is any other type of TP).

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70003I **TP name *tp_name* specified on inbound allocate request is not valid. It is the name of a SNA service TP that APPC/MVS does not support.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the TP name is not valid. It is the name of a SNA service TP that APPC/MVS does not support.

In the message text:

tp_name

The TP name specified in the side information or on the TP_name parameter on the call to Allocate.

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70005I TP name *tp_name* is not valid. An inbound allocate request specified a TP name with an incorrect character. If the TP is a SNA service TP, the first character is incorrect.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the TP name contains:

- One or more characters that are not from character set 00640 or the Type A character set, or
- A SNA service TP name that is not valid (because the first character is incorrect), or
- The customer has a client/server application in which the server TP name contains # (X'7B') or @ (X'7C').

In the message text:

tp_name

The name specified on the TP_name parameter on the call to Allocate.

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70006I TP name *tp_name* specified on the inbound allocate request is not valid. A request to allocate a SNA service TP specified an incorrect TP name.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the TP name contains one or more characters (besides the first character) that are not from the type A character set.

In the message text:

tp_name

The name specified on the TP_name parameter on the call to Allocate.

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70007I TP name *tp_name* specified on the inbound allocate request is not valid. The system could not find the associated TP profile.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The field in the FMH-5 that specifies the

TP name is not valid. The system could not find the associated TP profile for one of the following reasons:

- The profile does not exist
- *TP_name* is an alias that does not have an associated TP profile
- The allocate request specified *security_none*, and the profile is at a group or user level (when specifying *security_none*, the TP profile must be at a system level)
- The LU on which the request arrived has a TP level of SYSTEM, and the profile has a level of GROUP or USER (when the TP level is SYSTEM, only SYSTEM TP profiles are searched)
- The LU on which the request arrived has a TP level of GROUP, and the profile has a level of USER (when the TP level is GROUP, only group and system level TP profiles are searched)

In the message text:

tp_name

The TP name specified in the side information or on the *TP_name* parameter on the call to Allocate.

System action

APPC/MVS deallocates the conversation with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the request. Provide the sense code and the message explanation. Ask the support center to verify that the user's TP profile exists and is active, and that the TP level is correct.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70008I **TP security violation. Signed-on-from and signed-on-to lists are not synchronized.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A security violation occurred. The FMH-5 contains security information that is not synchronized

with the security information for the security product installed on MVS.

System action

The system sends an allocate request for the sign off TP (X'06F3F0F0') to the partner LU (to sign off the user). The system rejects the request with a sense code of *security_violation* (X'080F6051').

System programmer response

Contact the IBM Support Center.

Programmer response

Try to allocate the conversation again. If the problem persists, contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70009I **TP security violation. Inbound request to attach SIGNON SNA service TP specified a security_type that was not security_none.**

Explanation

APPC/MVS received an inbound request (FMH-5) to allocate a conversation with the SIGNON SNA service TP. The FMH-5 specified a security type value that is not valid. TPs can allocate the SIGNON TP only with a *security_type* of *security_none*.

System action

The system rejects the request with a sense code of *security_violation* (X'080F6051').

Programmer response

Change the value on the *security_type* parameter to 100 (*security_none*).

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70010I	TP security violation. Inbound allocate request specified a password, profile, or both, but did not specify a user ID.
------------------	---

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The FMH-5 did not specify a user ID, but did specify one or both of the following:

- A password
- A profile.

If the FMH-5 contains a password or a profile, the request must also specify a user ID.

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid user ID on the User_ID parameter.

Source

APPC/MVS

Detecting Module: ATBFMFP
--

ATB70011I	TP security violation. Allocate request specified a user ID but did not specify a password.
------------------	--

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The FMH-5 specified a user ID, but did not specify a password. A password is required if a user ID is specified, unless the request specifies a conversation security level of ALREADYV (for *already verified*) or PERSISTV (for *persistent verification*).

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid password on the Password parameter.

Source

APPC/MVS

Detecting Module: ATBFMFP
--

ATB70012I	TP security violation. Inbound allocate request specified persistent verification security level, but did not specify a user ID.
------------------	---

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The FMH-5 did not specify a user ID. The current security level supports persistent verification, which requires that the FMH-5 specify a user ID.

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid user ID on the User_ID parameter.

Source

APPC/MVS

Detecting Module: ATBFMFP
--

ATB70013I	TP security violation. Inbound allocate request specified 'already verified' security level, but did not specify a user ID.
------------------	--

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The inbound allocate request is already verified by the partner LU, but the FMH-5 did not specify a user ID. Requests with a security level of "already verified" must specify a user ID.

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid user ID on the User_ID parameter.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70014I **TP security violation. User ID *userid* is longer than the maximum number of characters allowed on MVS.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The FMH-5 specified a user ID that is longer than the maximum number of characters allowed on MVS (eight characters).

In the message text:

userid

The user ID specified in the FMH-5 used to pass the allocate request to MVS.

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid user ID on the User_ID parameter.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70015I **TP security violation. Password is longer than the maximum number of characters allowed on MVS.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The FMH-5 specified a password that is longer than the

maximum number of characters allowed on MVS (eight characters).

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid password on the Password parameter.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70016I **TP security violation. Profile name is longer than the maximum number of characters allowed on MVS.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The FMH-5 specified a profile name that is longer than the maximum number of characters allowed on MVS (eight characters).

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

Programmer response

On the call to the Allocate service, specify a valid profile name on the profile parameter.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70017I **TP security violation. Partner LU *plu_name* rejected the allocate request because authorization checks failed.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The TP encountered one of the following security violations, depending on the security_type value specified on the allocate request:

- The user ID, password, or optional group profile name is not valid
- The MVS LU does not accept security_none requests
- The user is not authorized to access the partner LU from this local LU.

In the message text:

plu_name

The name of the LU where the allocate request arrived.

System action

The system rejects the request with a sense code of security_violation (X'080F6051').

System programmer response

Depending on the security_type value specified on the allocate request, do one of the following:

- For a security_type of security_none, verify that the LU accepts security_none allocate requests by checking the security product class profiles that control user access to the LU, or checking the SECACPT parameter value on the VTAM APPL definition of the LU.
- For a security_type of security_pgm or security_same, verify that the user has authority to access the partner LU from this local LU by checking the security product class profiles that control user access **to** and **from** the LU, or by checking the SECACPT parameter value on the VTAM APPL definition of the LU.

See *z/OS MVS Planning: APPC/MVS Management* for more information about controlling access to or from LUs.

Programmer response

If a security_type of security_pgm was specified on the allocate request, verify the user ID, password, and group profile name (if any) that are specified on the allocate request. If these values are correct, or if a different security_type was specified on the allocate request, ask the security administrator to help you determine what caused the security violation.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70018I **TP security violation. Password specified for user *userid* has expired.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The user does not have access to the LU specified in the FMH-5. The specified password has expired.

In the message text:

userid

The user ID specified in the FMH-5 used to pass the allocate request to MVS. Blanks appear in this field if the specified Security_type is Security_none.

System action

The system rejects the request with a sense code of security_violation (X'080F6051'). The system sends to the partner LU an allocate request for the expired password notification TP (X'30F0F5F2') to tell the user that the password has expired.

Programmer response

Contact the system administrator to verify the required security information.

System Administrator Response: See the documentation for the security product installed on the other system for information about how to change the password.

Note: For information on setting up an IBM-supplied sample program to change passwords, see *z/OS MVS Planning: APPC/MVS Management*.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70020I **TP security violation. User *userid* not authorized to access TP *tp_name*.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). A TP security violation occurred. The user does not have authority to access the TP Profile specified in the FMH-5.

In the message text:

userid

The user ID specified in the FMH-5 used to pass the allocate request to MVS. Blanks appear in this field if the specified Security_type is Security_none.

tp_name

The TP name specified in the FMH-5 used to pass the allocate request to MVS.

System action

The system rejects the request with a sense code of security_violation (X'080F6051') or TP_access_denied (X'080F0983').

Programmer response

Contact the system administrator for the partner system to verify the required security information.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70021I TP security violation. User not authorized to test TP tp_name.

Explanation

APPC/MVS received an inbound request (FMH-5) to allocate a conversation with a TP that is registered for testing. APPC/MVS rejected the request because incorrect security information was specified in the FMH-5.

In the message text:

tp_name

The TP name specified in the FMH-5 used to pass the allocate request to MVS.

System action

The system rejects the request with a sense code of security_violation (X'080F6051') or TP_access_denied (X'080F0983').

Programmer response

Ensure that the security information specified on the allocate request is correct. If the information is correct, ask the partner system administrator to verify that the user has the authority to test the TP.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70022I TP security violation. User userid not authorized to run TP tp_name, which is served by an APPC/MVS server.

Explanation

APPC/MVS received an inbound request (FMH-5) to allocate a conversation with a TP that is served by an APPC/MVS server. A TP security violation occurred. The user who submitted the request is not authorized to allocate a conversation with this TP name to an APPC/MVS server.

In the message text:

userid

The user ID specified in the FMH-5 used to pass the allocate request to MVS. Blanks appear in this field if the specified Security_type is Security_none.

tp_name

The TP name specified in the FMH-5 used to pass the allocate request to MVS.

System action

The system rejects the request with a sense code of security_violation (X'080F6051') or TP_access_denied (X'080F0983'). The system does not queue the allocate request for a server.

System programmer response

At the request of the application programmer, see the section on installing APPC/MVS servers in *z/OS MVS Programming: Writing Servers for APPC/MVS* for information about the security profiles that protect the server. Provide the application programmer with the security information and, if necessary, ask the security administrator to grant the user access to the server.

Programmer response

Contact the system programmer to determine the required security information for the TP.

Security Administrator Response: At the request of the system programmer, update the security profile for the APPC/MVS server to allow the user to access the server.

Source

APPC/MVS

Detecting Module: ATBFMFP	
ATB70023I	APPC/MVS cannot process allocate request. PIP data is specified but is not supported by APPC/MVS.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The FMH-5 specified data in the PIP data area. APPC/MVS rejected the request because it cannot receive PIP data.

System action

The system rejects the request with a sense code of pip_not_allowed (X'10086031').

Programmer response

When allocating the conversation, specify a value of 0 on the pip_data_length parameter (to indicate that there is no PIP data to send).

Source

APPC/MVS

Detecting Module: ATBFMFP	
ATB70024I	APPC/MVS cannot process allocate request. Value in FMH-5 command field is not supported by APPC/MVS.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process a command that it found in the FMH-5.

System action

The system rejects the request with a sense code of command_not_valid (X'1008600B').

Programmer response

Contact the system programmer for the partner system. Ask the system programmer to correct the command in the FMH-5.

Source

APPC/MVS

Detecting Module: ATBFMFP	
ATB70025I	APPC/MVS cannot schedule an allocate request. LU <i>LU_name</i>, where the request arrived, is a NOSCHED LU.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The request arrived on an LU that is not associated with a transaction scheduler and has no APPC/MVS servers registered for the specified LU name (a NOSCHED LU) and the requested TP name.

In the message text:

LU_name

The name of LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

Operator response

At the request of the application programmer, enter a DISPLAY APPC command to verify that the LU is active and associated with a scheduler. If not, ask the application programmer to select another LU.

Programmer response

If the request is for an APPC/MVS server, then ask the operator to verify that a server has registered with the LU name, TP name and optionally other filters. Note that TP name is case sensitive. Otherwise, ask the operator to verify that the LU is active and associated with a scheduler. If not, specify the name of another LU that is active and associated with a scheduler.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70026I **APPC/MVS cannot schedule an allocate request. Scheduler *sched_name*, associated with LU *LU_name* where the request arrived, is not active.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. The transaction scheduler that is associated with the LU on which the allocate request arrived is not active.

The following is an example of a situation that can cause the error:

- The conversation is LU=OWN
- The LU on which the request arrived is the ASCH base LU
- There is no NOSCHED base LU (which means the ASCH base LU is also the system base LU).

In the message text:

sched_name

The name of the transaction scheduler associated with the LU specified in the message text.

LU_name

The name of LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

Operator response

At the request of the application programmer, start the transaction scheduler identified in the message text.

Programmer response

Ask the operator to start the transaction scheduler.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70027I **APPC/MVS cannot schedule allocate request. LU *LU_name* is not in active state.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The status of the LU on which the request arrived is not "active".

In the message text:

LU_name

The name of LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

At the request of the application programmer, see [z/OS Communications Server: SNA Network Implementation Guide](#) for information about how to diagnose a problem with the VTAM definition for the LU.

Programmer response

Verify that the call to the Allocate service specifies a valid LU name on the Partner_LU_name parameter. If the LU name is correct, ask the system programmer to verify that the LU is defined to VTAM.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70028I **APPC/MVS cannot process allocate request. LU *LU_name* is not defined to APPC/MVS.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). The LU that was supposed to receive the request is either not active or it is not defined to APPC/MVS.

In the message text:

LU_name

The name of LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

Operator response

At the request of the application programmer, enter a DISPLAY APPC command to verify that the LU is defined to APPC/MVS. At the request of the system programmer, enter a SET APPC command to activate the LU. If the LU does not become active when you enter the SET APPC command, validate that VTAM is active.

System programmer response

At the request of the application programmer, define the LU in an APPCPMxx parmlib member. If necessary, see the section on controlling configuration in [z/OS MVS Planning: APPC/MVS Management](#) for more information about defining local LUs. Then ask the operator to enter a SET APPC command to activate the LU in the APPC/MVS configuration.

Programmer response

Verify that call to the Allocate service specified a valid LU name on the Local_LU_name parameter. If the LU name is correct, ask the operator to verify that the LU is defined to APPC/MVS. If the LU is defined, contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70030I **APPC/MVS cannot schedule TP tp_name. The TP profile is not active.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS cannot schedule the TP. The TP profile is not active.

In the message text:

tp_name

The TP name specified in the FMH-5 used to pass the allocate request to MVS.

System action

The system rejects the request with a sense code of TP_not_available_retry (X'084B6031').

System programmer response

At the request of the application programmer, set the ACTIVE parameter in the TP profile to YES to activate the TP profile.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70031I **APPC/MVS cannot schedule allocate request. TP profile for requested TP is an alias of another TP profile, which is an alias of a third TP profile. A TP profile cannot be an alias of another alias TP profile.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS cannot schedule the request. The TP profile for the requested TP is an alias for another TP profile, which is an alias for a *third* TP profile. The TP profile for the requested TP cannot be an alias of another TP profile that is an alias of a third TP profile.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Enter a TPALIAS command to add a valid alias to the VSAM file that contains the TP profile information. Enter a TPDELETE command to delete the TP profile that had an incorrect alias. See the section on using the APPC/MVS administration utility in [z/OS MVS Planning: APPC/MVS Management](#) for more information about adding a profile alias to a TP profile.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70032I APPC/MVS cannot schedule an allocate request. A syntax error was found in the TP profile.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS cannot schedule the request. APPC/MVS found the profile for the requested TP, but the profile contained:

- A syntax error, or
- A data set name that used the &SYSUID variable when the requested level of security was security_none.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that issued the allocate request.

System programmer response

At the request of the application programmer, do one of the following:

- Correct the syntax in the TP profile
- Remove the &SYSUID variable or specify a level of security other than security_none.

See *z/OS MVS Planning: APPC/MVS Management* for more information about modifying the contents of a TP profile.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70033I APPC/MVS cannot schedule an allocate request. Scheduler sched_name, associated with LU LU_name where the request arrived, is not a member of the APPC XCF group. The scheduler can not be notified about the inbound allocate request.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. The transaction scheduler that is associated with the LU on which the allocate request arrived is not an active member of the APPC/MVS XCF group. XCF could not deliver the inbound allocate request message to the transaction scheduler message user routine.

In the message text:

sched_name

The name of the transaction scheduler associated with the LU specified in the message text.

LU_name

The name of the LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

System programmer response

Determine the reason why the transaction scheduler on the target system is not an active member of the APPC XCF Group, but is still identified to APPC/MVS as a transaction scheduler. A transaction scheduler should not remain identified to APPC/MVS when the transaction scheduler is not an active member of the APPC XCF group.

Programmer response

Ask the system programmer to contact the support center for the system that rejected the FMH-5 request.

Source

APPC/MVS

Detecting Module

ATBFMFP

ATB70034I APPC/MVS cannot schedule an allocate request. Scheduler

sched_name, associated with LU ***LU_name*** where the request arrived, did not specify a valid XCF message user routine. The scheduler can not be notified about the inbound allocate request.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. The transaction scheduler that is associated with the LU on which the allocate request arrived is not an active member of the APPC/MVS XCF group. The Inbound Allocate Request XCF message could not be delivered because the transaction scheduler did not specify an XCF message user routine on a Join_Sysappc_Group service call or IXCJOIN macro call when joining the APPC XCF group.

In the message text:

sched_name

The name of the transaction scheduler associated with the LU specified in the message text.

LU_name

The name of the LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

System programmer response

Determine the reason why the transaction scheduler on the target system did not provide an XCF message user routine when joining the APPC XCF group.

Programmer response

Ask the system programmer to contact the support center for the system that rejected the FMH-5 request.

Source

APPC/MVS

Detecting Module

ATBFMFP

ATB70034I	APPC/MVS cannot schedule an allocate request. Scheduler <i>sched_name</i>, associated with LU <i>LU_name</i> where the request
------------------	---

arrived, did not specify a valid XCF message user routine. The scheduler can not be notified about the inbound allocate request.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. The transaction scheduler that is associated with the LU on which the allocate request arrived could not be notified of an Inbound Allocate Request. The Inbound Allocate Request XCF message could not be delivered because the transaction scheduler did not specify an XCF message user routine on a Join_Sysappc_Group service call or IXCJOIN macro call when joining the APPC XCF Group.

In the message text:

sched_name

The name of the transaction scheduler associated with the LU specified in the message text.

LU_name

The name of the LU on which the allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

System programmer response

Determine the reason why the transaction scheduler on the target system did not provide an XCF message user routine when joining the APPC XCF group.

Programmer response

Ask the system programmer to contact the support center for the system that rejected the FMH-5 request.

Source

APPC/MVS

Detecting Module

ATBFMFP

ATB70035I	APPC/MVS cannot schedule an allocate request. XCF Signalling Services could not deliver an inbound allocate request message to a scheduler XCF message user
------------------	--

routine due to the unavailability of message buffers.

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. APPC/MVS was attempting to send an Inbound Allocate Request message to a transaction scheduler, which is a member of the APPC/MVS XCF group. The attempt failed because XCF Signalling Services message buffer space was not available after numerous retry attempts. Some of the reasons message buffer space might not be available are:

- There was suddenly a large amount of message buffer space usage, which caused all the buffer space to be temporarily exhausted.
- There is a lot of competition for message buffer space. It is possible that the installation should have allocated more message buffers for the transport class used by the APPC XCF group. The installation can use the message buffer limit to control how much of the total message buffer resource can be used by a specific transport class.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

System programmer response

Modify the amount of message buffer space made available for local message traffic for the transport class used by the APPC XCF Group on the local system. For more information about modifying the amount of message buffer space made available for local XCF message traffic, see the SETXCF MODIFY command in [z/OS MVS System Commands](#).

Programmer response

Ask the system programmer to evaluate the message buffer space usage for the transport class used by the APPC XCF group.

Source

APPC/MVS

Detecting Module

ATBMIMO

ATB70036I **APPC/MVS cannot schedule an allocate request. XCF Signalling**

Services could not deliver an inbound allocate request message to a scheduler XCF message user routine. XCF IXCMMSGO Return Code: xxxxxxxx XCF IXCMMSGO Reason Code: yyyyyyyy

Explanation

APPC/MVS received an inbound allocate request (FMH-5). APPC/MVS could not process the request. APPC/MVS was attempting to send an Inbound Allocate Request message to a transaction scheduler, which is a member of the APPC/MVS XCF group. The attempt failed due to a failure of the XCF Signalling Services. The return and reason code from the XCF Signalling Services IXCMMSGO macro are supplied in the message.

In the message text:

xxxxxxx

The return code from the XCF Signalling Services IXCMMSGO macro.

yyyyyyy

The reason code from the XCF Signalling Services IXCMMSGO macro.

Note: If the message indicates return code 8 and reason code C for the IXCMMSGO macro, ensure that the userid being passed does not have too large a security environment definition, because the security information for the userid is part of the message that APPC passes with the IXCMMSGO call.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000') to the TP that called the allocate service.

System programmer response

Determine the reason for the XCF failure. The service return and reason codes explain the error. For the description of the return and reason codes, see *z/OS MVS Programming: Sysplex Services Reference*. Correct the problem and have the application programmer send the allocate request again.

Programmer response

Ask the system programmer to contact the support center for the system that rejected the FMH-5 request.

Source

APPC/MVS

Detecting Module

ATBFMFP, ATBMIMO

ATB70040I **APPC/MVS cannot schedule allocate request. Log name for LU *lu_name* is unknown.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5) for an conversation with synchronization level of syncpt (protected conversation). APPC/MVS cannot process the inbound allocate request because an exchange log name transaction is required to complete successfully between the APPC/MVS logical unit and the partner logical unit prior to initiating or receiving allocate requests for syncpoint conversation by either logical unit.

An exchange log name transaction may have completed successfully between the logical unit pairs previously, but if all sessions between the logical units are lost, another exchange log name request must be initiated to confirm that the log names and syncpoint capabilities negotiated between the logical units have not changed.

In the message text:

lu_name

The name of the LU that initiated the allocate request.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

The logical unit (LU) that is initiating the Allocate Attach request (sending the FMH-5) should ensure that a successful exchange log name transaction completes prior to sending of an Allocate Attach request for a SYNCPT conversation. The sending LU should initiate an exchange log name transaction with the APPC/MVS LU that sent this message.

If an exchange log name transaction had already completed between the partner LUs, the VTAM VARY TERM command can be issued to terminate all of the sessions between a specified pair of logical units. All sessions, including the SNASVCMG sessions, must terminate in order to cause the APPC/MVS LU to initiate an exchange log name transaction with the LU that received this message. The VARY TERM command terminates all sessions abnormally.

Optionally the MODIFY CNOS command can be issued to quiesce sessions between a specified pair of logical

units without abnormally terminating conversations. Once the last session between the LU pair has been deactivated, APPC/MVS will initiate an exchange log name transaction. After the transaction completes, Allocate Attach requests for syncpt conversations will be accepted by the APPC/MVS LU.

For more information on the VTAM MODIFY and VARY commands, see [*z/OS Communications Server: SNA Operation*](#).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70041I **APPC/MVS cannot schedule allocate request. LU *lu_name* is not syncpt capable.**

Explanation

APPC/MVS received an inbound Allocate request (FMH-5) to initiate exchange log-name processing for support of protected conversations (that is, conversations with a synchronization level of syncpt). APPC/MVS cannot process the inbound Allocate request because the local LU does not support the receipt of Allocate requests for protected conversations.

In the message text:

lu_name

The name of the LU on which the Allocate request arrived.

System action

The system rejects the request with a sense code of TPN_not_recognized (X'10086021').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70042I **APPC/MVS cannot schedule allocate request. LU *lu_name* cannot process syncpt conversations.**

Explanation

APPC/MVS received an inbound Allocate request (FMH-5) for either:

- A protected conversation (a conversation with a synchronization level of syncpt), or
- An inbound Allocate request for a SNA Service TP to perform processing associated with protected conversation support (that is, a request for an exchange log-name transaction).

APPC/MVS cannot process the inbound Allocate request because the LU's resource manager exits are not set with the system syncpoint manager (RRS) for one of the following reasons:

- The system syncpoint manager is not active.
- An error that occurred during resource manager restart processing has prevented the local LU from registering as a resource manager with the system syncpoint manager.

In the message text:

lu_name

The name of the LU on which the Allocate request arrived.

System action

The system rejects the request with a sense code of TP_not_available_retry (X'084B6031').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70043I **APPC/MVS cannot process allocate request. Sync_level specified for a RESYNC TP is not CONFIRM.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5) to initiate exchange log name processing for support of syncpt (protected) conversations. APPC/MVS cannot process the inbound allocate request because the FMH-5 for the attach request contains a synchronization level other than confirm.

System action

The system rejects the request with a sense code of Sync_Lvl_Not_Supported (X'10086041').

System programmer response

Contact the support center for the system that rejected the FMH-5 request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70044I **APPC/MVS cannot process allocate request. Conversation type specified for a RESYNC TP is not BASIC.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5) to initiate exchange log name processing for support of syncpt (protected) conversations. APPC/MVS cannot process the inbound allocate request because the FMH-5 for the attach request contains a Conversation Type other than Basic.

System action

The system rejects the request with a sense code of Conversation_Type_Mismatch (X'10086034').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

**ATB70050I Receive allocate request failed:
Return code: *return_code***

Explanation

APPC/MVS received an inbound request (FMH-5) to allocate a syncpt conversation with a TP that is served by an APPC/MVS server. During the processing to receive the allocate request, an error occurred when the APPC/MVS Receive_Allocate service was attempting to interface with the system syncpoint manager (RRS/MVS). The allocate was not received by the server TP.

In the message text:

return_code

The return code value from an RRS/MVS service that failed, causing the receive allocate request to fail and be rejected by APPC/MVS.

System action

The system rejects the request with a sense code of TP_not_available_retry (X'084B6031').

System programmer response

Contact the support center for the system that rejected the allocate request. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBVSPA

**ATB70051I APPC/MVS detected a protocol
violation during an Exchange Log
Name processing. Reason Code:
*reason_code***

Explanation

During exchange-log-name processing, an LU detected an error in the data sent by its partner LU. A reason code further explains the error by identifying the protocol violation. Message ATB206E or ATB218E is also issued when this error occurs, and identifies the local and partner LUs.

In the message text, *reason_code* is one of the following:

01

EXCHANGE LOG NAME GDS VARIABLE FORMAT ERROR: An exchange log name GDS variable received in reply to an exchange log name request initiated by the local LU contains a format error.

04

EXCHANGE LOG NAME GDS VARIABLE FORMAT ERROR: An exchange log name GDS variable received as part of an exchange log name request initiated by the partner LU contains a format error.

05

UNEXPECTED DATA RECEIVED FROM INITIATOR: Unexpected data was received from a partner who was initiating a cold-start exchange log name transaction.

06

DEALLOCATE ABEND OF CONVERSATION NOT RECEIVED: A deallocation of the exchange logname or resynchronization transaction conversation from the initiator was expected, but not received.

07

UNEXPECTED STATUS DATA RECEIVED FROM PARTNER: Unexpected status data was received from a partner who was replying to an exchange log name or resynchronization transaction initiated by the local LU.

08

NO DATA RECEIVED FROM THE PARTNER: During a resynchronization or exchange log name transaction exchange, the partner responded but failed to send GDS variable data containing the state of the partner LU.

09

UNEXPECTED DATA RECEIVED FROM PARTNER: Unexpected data was received from a partner who was replying to an exchange log name or resynchronization transaction initiated by the local LU.

10

INVALID STATUS DATA RECEIVED FROM THE PARTNER: Status data that was invalid for the reply was received by the initiator of the exchange log name or resynchronization transaction.

11

NO DATA RECEIVED FROM THE INITIATOR: The initiator of the SNA service TP request failed to send GDS variable data describing the request.

12

TOO MUCH DATA RECEIVED FROM THE INITIATOR: The initiator of the SNA service TP request sent more than the expected amount of GDS variable data for the request.

13

INVALID STATUS DATA RECEIVED FROM THE INITIATOR: Status data that was invalid for the request was received by the partner of the exchange log name or resynchronization transaction.

16

SYNCPT CAPABILITIES NEGOTIATION NOT ALLOWED: The partner attempted to negotiate syncpt capabilities while there was outstanding resynchronization work to be performed between the local and partner LUs.

18

SYNCPT CAPABILITIES DO NOT MATCH: The syncpt capabilities sent in an exchange log name GDS variable for a warm-start exchange do not match the capabilities previously negotiated by the local and partner LUs.

System action

If this message is issued by APPC/MVS resynchronization processing, resynchronization does not complete successfully. If this message is issued during an exchange log name interchange preceding a protected conversation allocate or inbound attach request, the protected conversation between the local and partner LU is not allocated. No protected conversations between the local and partner LU will be allocated until the protocol violation can be resolved.

The LU that made the protocol violation receives this message as log data; the LU that detected the error may have written additional diagnostic information into its system log to identify the violation made by the recipient of this message.

System programmer response

Examine the log of the partner LU's system. If a protocol violation was detected in the local system's Exchange Log Names GDS variable, the remote system may have generated diagnostic information itself. This information may help to diagnose the cause of a protocol violation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB70052I

APPC/MVS detected a warm/cold log status mismatch during an Exchange Log Name processing.

Explanation

This message is issued during an exchange log name transaction when the partner LU has detected a warm/cold log status mismatch. Message ATB210E is issued on the system that detected the mismatch.

The LU that has the cold status receives this message as log data. The LU that detected the error writes this message into the logrec data set, with additional diagnostic information to assist in diagnosing the problem.

System action

If this message is issued by APPC/MVS resynchronization processing, resynchronization does not continue. Resynchronization will automatically be attempted again at a later time. If this message is issued during an exchange log name interchange preceding a protected conversation allocate or inbound attach request, the protected conversation between the local and partner LU is not allocated. No protected conversations between the local and partner LU will be allocated until the warm/cold mismatch can be resolved.

Additional diagnostic information is written into the logrec data set.

System programmer response

To resolve the warm/cold mismatch, see z/OS MVS Planning: APPC/MVS Management.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB70053I **APPC/MVS detected a log name mismatch during an Exchange Log Name processing.**

Explanation

This message is issued during an exchange log name transaction when the partner LU has detected a log name mismatch. Message ATB211E is issued on the system that detected the log name mismatch.

The LU that provided the reply containing the conflicting log name receives this message as log data; the LU that detected the error writes this message into the logrec data set, with additional diagnostic information to assist in diagnosing the problem.

System action

If this message is issued during APPC/MVS resynchronization processing to resolve incomplete units of recovery, resynchronization does not continue. Resynchronization will be attempted again automatically at a later time.

If this message is issued during an exchange log name interchange preceding a protected conversation allocate or inbound attach request, the protected conversation between the local and partner LU is not allocated. No protected conversations between the local and partner LU will be allocated until the warm/cold mismatch can be resolved.

Symptom records are written to the logrec data set to record the error condition and record diagnostic data.

System programmer response

To resolve the warm/cold mismatch, z/OS MVS Planning: APPC/MVS Management.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB70054I **Conversation was terminated because a protocol violation was detected by LU *partner_lu* during a syncpoint processing. Reason code = *rsncode intrsncode*.**

Explanation

The partner LU has detected a response sent by the local LU that violates the syncpoint exchange protocol during the syncpoint processing of a logical unit of work. Message ATB220I is issued for this error condition on the partner LU's system, and identifies the local LU and the logical unit of work identifier.

In the message text:

partner_lu

The network-qualified name of the partner LU that detected the protocol violation

rsncode

One of the following:

01

NO PS HEADER WAS RECEIVED

02

EXPECTED PS HEADER WAS NOT RECEIVED

03

EXPECTED STATUS WAS NOT RECEIVED

04

UNEXPECTED RETURN CODE WAS RECEIVED

05

UNEXPECTED DATA WAS RECEIVED

06

CONVERSATION STATE WAS INVALID

intrsncode

Information for IBM use only.

System action

The syncpoint processing continues, but the protected conversation is deallocated by the partner LU that detected the protocol violation, and the state of the distributed resources is unknown; a heuristic condition might exist.

The LU that made the protocol violation receives message ATB70054I as log data.

System programmer response

Contact the designated support group for your installation.

Programmer response

Notify the system programmer.

Source

APPC/MVS

Detecting Module:

**ATBPCPR, ATBPCBO, ATBPCDS, ATBPCCM,
ATBPCEU, ATBPCEE**

**ATB70055I Conversation was terminated due
to a break-tree condition or a
terminating syncpoint situation.**

Explanation

This message is sent as log data when a conversation is deallocated due to a break-tree condition or when the last syncpoint (terminating synchronization point) for a unit of work completes. For an APPC/MVS TP issuing Error_Extract, this information is presented to a TP for a conversational verb that allows a deallocated_abend_* return code to be presented.

System action

The called service returns a return code of deallocated_abend_svc (decimal 30). The conversation has been deallocated and all resources associated with the conversation have been cleaned up.

Source

APPC/MVS

Detecting Module:

ATBPCEE

**ATB70056I APPC/MVS detected a
protocol violation during a
Resynchronization Exchange.
Reason Code: *rsncode*.**

Explanation

During the processing of a resynchronization transaction request, an LU detected an error in the data sent by its partner LU. A reason code further explains the error by identifying the protocol violation. Message ATB206E or ATB218E is also issued on the detecting system when this error occurs, and identifies the local and partner LUs.

In the message text, *rsncode* is one of the following:

01

**EXCHANGE LOG NAME GDS VARIABLE FORMAT
ERROR**

An Exchange Log Name GDS variable received in reply to an exchange log name request initiated by the local LU contains a format error.

02

COMPARE STATES GDS VARIABLE NOT RECEIVED

During a resynchronization exchange, the partner did not send a Compare States GDS variable reply containing the state of the logical unit of work at the partner LU.

03

COMPARE STATES GDS VARIABLE FORMAT ERROR

A Compare States GDS variable received in reply to a resynchronization request initiated by the local LU contains a format error.

14

COMPARE STATES GDS VARIABLE FORMAT ERROR

A Compare States GDS variable received as part of a resynchronization request initiated by the partner LU contains a format error.

System action

This message is issued by APPC/MVS resynchronization processing. Resynchronization does not complete for an incomplete unit of recovery.

The LU that made the protocol violation receives this message as log data; the LU that detected the error may have written additional diagnostic information into its system log to identify the violation made by the recipient of this message.

System programmer response

Examine the log of the partner LU's system. If a protocol violation was detected in the local system's Exchange Log Names GDS variable or Compare States GDS variable, the remote system may have generated diagnostic information itself. This information may help to diagnose the cause of a protocol violation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB70057I **APPC/MVS detected an error during Purge Log Name TP processing. Reason Code: *rsncode*.**

Explanation

A purge log name affinity (PLNA) request either initiated or received by an APPC/MVS logical unit failed.

In the message text:

rsncode

One of the following:

- 1** Reply format error
- 2** Request format error
- 5** No data received from partner
- 6** Unexpected data received from partner
- 7** Invalid status received from partner
- 8** No data received from initiator
- 9** Too many buffers received from initiator
- 10** Invalid status received from initiator

Other values

Internal reason codes for IBM use only.

Detecting Module:

ATBPCPL

System action

A purge log name affinity (PLNA) request failed. The system continues processing, but log name affinities between an APPC/MVS logical unit and a partner LU persist. The system on which the error was incurred writes this message to the logrec data set, and sends the message to the partner purge-log-name TP.

System programmer response

Use the diagnostic records written to the logrec data set to identify the reason for the failure.

Note: When internal reason codes are issued, report the complete text of this message to your IBM support center.

ATB70058I **APPC/MVS cannot process allocate request. Syncpoint**

conversations are not supported when the partner LU session capabilities are single session.

Explanation

APPC/MVS received an inbound allocate request (FMH-5) for an conversation with synchronization level of syncpt (protected conversation). APPC/MVS cannot process the inbound allocate request because APPC/MVS does not support allocating syncpt conversations with a partner LU that does not have parallel session capability.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70059I **APPC/MVS cannot process allocate request. Syncpoint conversations are not supported on the SNASVCMG session.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5) for an conversation with synchronization level of syncpt (protected conversation). APPC/MVS does not accept inbound allocate requests for syncpt conversations on a session associated with the SNASVCMG mode name.

System action

The system rejects the request with a sense code of TP_not_available_retry (X'084B6031').

Programmer response

Allocate the syncpt conversation with a mode name other than SNASVCMG.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70061I **APPC/MVS cannot process allocate request. Conversation type specified for a PLNA TP is not BASIC.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5) to initiate purge log name affinities associated with the support of syncpt (protected) conversations. APPC/MVS cannot process the inbound allocate request because the FMH-5 for the attach request contains a Conversation Type other than Basic.

System action

The system rejects the request with a sense code of Conversation_Type_Mismatch (X'10086034').

System programmer response

Contact the support center for the system that sent the FMH-5. Provide the sense code and the message explanation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBFMFP

ATB70999I **An APPC/MVS internal error occurred: Reason code: reason_code.**

Explanation

APPC/MVS received an inbound allocate request (FMH-5). An internal error occurred while APPC/MVS was processing the request.

In the message text:

reason_code

The internal reason code (in hexadecimal), which is useful to the IBM Support Center when diagnosing the error.

System action

The system rejects the request with a sense code of TP_not_available_no_retry (X'084C0000').

System programmer response

Contact the IBM Support Center. Provide the reason code shown in the message text.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

All

Error_Extract (ATB8) Messages

The Error_Extract service can return one of the messages described in this section when APPC/MVS finds an error in the most recently completed call to another APPC TP conversation service or CPI-C call. These messages appear on the Message_text parameter on return from Error_Extract.

Error_Extract also returns error log information messages when a partner system or TP finds an error in a call to an APPC TP conversation service or CPI-C call. For descriptions of those error log information messages, see [“Error_Extract Error Log Information \(ASB, ATB7\) Messages” on page 317.](#)

ATB80003I **APPC data structures for the TP are in use by another process.**

Explanation

A TP called an APPC/MVS TP conversation service. The system could not process the request because

APPC/MVS data structures for the TP are in use by another process.

One of the following situations could cause this error:

- The TP called the TP conversation service while the system was processing a call to another conversation callable service for the same conversation. For example, the previous call could have specified a Notify_type of Notify_ecb, and the TP could have called the second service before the system could post the ECB (to indicate the end of asynchronous processing for the previous call).
- An internal error occurred in APPC/MVS.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

System programmer response

Contact the IBM Support Center.

Programmer response

Ensure that processing for a previous call to an APPC/MVS service is complete before you call another APPC/MVS service. If a Notify_type of Notify_ecb was specified on the previous call, wait on the ECB before calling the next service. If the problem persists, contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVS RB

ATB80004I **System cannot process a call. A Cleanup_TP request is in progress.**

Explanation

A TP issued an APPC/MVS TP conversation service while a Cleanup_TP request was in progress for the same TP. A TP cannot call another service while the system is processing a Cleanup_TP request for the same TP.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

Programmer response

Change the TP so it does not call a conversation service while the system is processing a Cleanup_TP request.

Source

APPC/MVS

Detecting Module:
ATBVS RB

ATB80005I **APPC/MVS could not retrieve side information.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. An error occurred when APPC/MVS read the side information entry for the symbolic destination name specified on the request.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate or Initialize_Conversation service. The system writes a logrec data set record that describes the error.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSIN

ATB80006I **Error retrieving security information.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. An error occurred when APPC/MVS tried to obtain information about the caller's security environment from the security product.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate service. APPC/MVS writes a logrec data set record that describes the error.

System programmer response

Validate that the correct level of the security product is installed and running. The return and reason codes from the security product appear in section 5 of the symptom record in the logrec data set. If your installation is using RACF, see *z/OS Security Server RACF Messages and Codes* for explanations of the return and reason codes.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSCA

ATB80008I Address space cannot use the system base LU.

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. The request did not specify a local LU name, so APPC/MVS selected a local LU to be the source of the conversation. The program is running in an address space that is associated with an alternate transaction scheduler. No base LU is defined for the alternate scheduler. The system base LU is owned by the APPC scheduler (ASCH), so the address space in which the TP is running cannot use the system base LU.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate or Initialize_Conversation service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine if other LUs are available to the scheduler.

At the request of the system programmer, enter a SET APPC=xx command to activate a newly defined LU in the APPC/MVS configuration.

System programmer response

At the request of the application programmer, enter an LUADD statement in an APPCPMxx parmlib member to define a base LU for the alternate transaction scheduler or a NOSCHED LU. Then ask the operator to enter a SET APPC=xx command to activate the newly defined LU in the APPC/MVS configuration.

Programmer response

Do one of the following:

- Contact the system programmer to determine if a base LU can be defined for the alternate transaction scheduler.
- Contact the system programmer to determine if a NOSCHED base LU can be defined.
- Contact the operator to determine if other LUs are available to the scheduler. These may be LUs associated with that scheduler, or LUs not associated with any scheduler (NOSCHED LUs). Change the TP to use the ATBALC2 version of Allocate, and specify the name of an LU available to the scheduler on the Local_LU parameter.
- Ensure that the TP tries to allocate or initialize the conversation from an address space that is able to use the system base LU.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSIN

ATB80009I No base LU defined for scheduler.

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. The request did not specify a local LU name, so APPC/MVS had to select a local LU as the source of the conversation. The TP is running in an address space that is associated with a scheduler. No base LU is defined for the transaction scheduler, and no system base LU is defined. Therefore, APPC/MVS could not determine which LU to use as the source of the conversation.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate or Initialize_Conversation service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine if other LUs are available to the scheduler.

At the request of the system programmer, enter a SET APPC=xx command to activate a newly defined LU in the APPC/MVS configuration.

System programmer response

At the request of the application programmer, enter an LUADD statement in an APPCPMxx parmlib member to define a base LU for the transaction scheduler or a NOSCHED LU. Then ask the operator to enter a SET APPC=xx command to activate the newly defined LU in the APPC/MVS configuration.

Programmer response

Do one of the following:

- Contact the system programmer to determine if a base LU can be defined for the transaction scheduler.
- Contact the system programmer to determine if a NOSCHED base LU can be defined.
- Contact the operator to determine if other LUs are available to the scheduler. These may be LUs associated with that scheduler, or LUs not associated with any scheduler (NOSCHED LUs). Change the TP to use the ATBALC2 (or higher) version of Allocate and specify the name of an LU available to the scheduler on the Local_LU parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSIN

ATB80010I Scheduler extract exit could not identify active TP.

Explanation

A TP called an APPC/MVS conversation service in an address space where more than one TP was running. APPC/MVS called the transaction scheduler extract exit to obtain the TP_ID for the active TP. The exit returned a non-zero return code to APPC/MVS.

System action

One of the following:

- If the TP called the Allocate or Initialize_Conversation service, the system returns a product_specific_error (decimal 20) return code to the caller, and APPC/MVS writes a symptom record in the logrec data set
- If the TP called the LU 6.2 Get_Conversation or CPI-C Accept_Conversation services, the system returns a program_state_check (decimal 25) return code to the caller, and APPC/MVS writes a symptom record in the logrec data set

System programmer response

See the symptom record in the logrec data set for a description of the error. Check the return code from the transaction scheduler extract exit in the Scheduler Extract Control Block (ATBSECB) in section 5 of the symptom record. The ATBSECB is in the first key-length-data structure in section 5. See *z/OS MVS Data Areas* in the *z/OS Internet library* (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a description of the ATBSECB.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSIN, ATBVSGC

ATB80011I Storage not available for APPC internal structures.

Explanation

A TP called an APPC/MVS conversation service. APPC/MVS could not obtain enough storage to process the request.

System action

The system returns a product_specific_error (decimal 20) or resource_failure_no_retry (decimal 26) return code to the caller of the conversation service. APPC/MVS writes a logrec data set error record that describes the error.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBAMIC, ATBAMLM, ATBAMLPL, ATBVSAL, ATBVSDE, ATBAMDE, ATBAMEL, ATBAMRT, ATBAMTS, ATBVSGC, ATBVSCD, ATBVSCF, ATBVSIN, ATBVSPT, ATBVSFL, ATBVSRC, ATBVSRT, ATBVSSD, ATBVSSR, ATBVSST, ATBVSRB

ATB80014I Service interrupted by call to Cleanup_TP service.

Explanation

A TP called an APPC/MVS conversation service. While APPC/MVS was processing the call, a program called the Cleanup_TP service to clean up resources for the TP that owns the conversation. APPC/MVS terminates all services in progress on all conversations for the TP before processing the Cleanup_TP request.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the interrupted conversation service.

Programmer response

Change the program so it does not call the Cleanup_TP service until all other calls to conversation services are complete.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSCF, ATBVSGA, ATBVSGC, ATBVSIN, ATBVSRC, ATBVSSR, ATBVSTR, ATBVS3XT, ATBVS4XT, ATBVSCA

ATB80016I An internal error occurred in APPC processing.

Explanation

A TP called an APPC/MVS conversation service. An internal error occurred in APPC processing.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service. APPC/MVS writes a logrec data set record that describes the error. The system might request an SVC dump.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record and the SVC dump (if one is available).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBVSRB, ATBAMAL, ATBAMDE, ATBAMLM, ATBAMRC, ATBAMRE, ATBAMSR, ATBVSRG, ATBVSRF, ATBVSRD, ATBVSSD, ATBVSRAL, ATBVSGC, ATBVSRIN, ATBAMPR

ATB80020I Information about local LU was not available to APPC/MVS.

Explanation

A TP called the LU 6.2 or CPI-C Allocate service. APPC/MVS could not find information about the local LU. An internal error prevented APPC/MVS from locating the LU.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate service. APPC/MVS writes a logrec data set record that describes the error. The system might request an SVC dump.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record and the SVC dump (if one is available).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBAMAL

**ATB80023I Processing for service interrupted
by Deallocate_abend.**

Explanation

A TP called an APPC/MVS conversation service. While the conversation service was in progress, the TP also called the Deallocate service with a Deallocate_type of Deallocate_abend (to deallocate the conversation abnormally). APPC/MVS abnormally ended the conversation and ended processing for the call to the conversation service. (This message is expected if you deallocate the conversation abnormally to deliberately interrupt processing for the service).

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the interrupted conversation service.

Programmer response

Change the TP so it does not call the Deallocate service with a Deallocate_type of Deallocate_abend while a call to another conversation service is in progress.

Source

APPC/MVS

Detecting Module:

**ATBAMAL, ATBAMCF, ATBAMCD, ATBAMDE,
ATBAMFL, ATBAMPT, ATBAMRC, ATBAMSD,
ATBAMSR, ATBAMRT, ATBAMLM**

**ATB80025I A previous error left the
conversation in an undefined
state.**

Explanation

A TP called an APPC/MVS conversation service. APPC/MVS cannot process the request because a

previous error left the conversation in an undefined state.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

Programmer response

Call the Deallocate service with a Deallocate_type of Deallocate_abend (to deallocate the conversation abnormally and free the resources for the conversation). APPC/MVS will end the current session.

Source

APPC/MVS

Detecting Module:

**ATBAMCF, ATBAMDE, ATBAMPT, ATBAMSD,
ATBAMSR**

**ATB80026I An unexpected resource failure
occurred.**

Explanation

A TP called an APPC/MVS conversation service. APPC/MVS found a resource failure while processing a service that does not return the resource_failure_no_retry (decimal 26) or resource_failure_retry (decimal 27) return codes.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service. The system writes a logrec data set record that describes the error.

System programmer response

Contact the IBM Support Center. Provide the logrec data set error record.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBAMLS

ATB80033I Address space in which TP is running cannot use system base LU.

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. The request did not specify a local LU name, so APPC/MVS had to select a local LU as the source of the conversation. A previous call to the Set_AS_Attributes service prohibited the address space in which the program is running from using the system base LU. Either the address space is not connected to a scheduler, or the address space is connected to a scheduler for which a base LU is not defined.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine if other LUs are available to the scheduler.

At the request of the system programmer, enter a SET APPC=xx command to activate a newly defined LU in the APPC/MVS configuration.

System programmer response

At the request of the application programmer, enter an LUADD statement in an APPCPMxx parmlib member to define a base LU for the alternate transaction scheduler or a NOSCHED LU. Then ask the operator to enter a SET APPC=xx command to activate the newly defined LU in the APPC/MVS configuration.

Programmer response

Do one of the following:

- If the address space is not connected to a scheduler, see if the transaction scheduler for the address space has terminated. If so, ask the operator to start the scheduler again.
- Contact the system programmer to determine if a base LU can be defined for the alternate transaction scheduler.
- Contact the system programmer to determine if a NOSCHED base LU can be defined.
- Contact the operator to determine if other LUs are available to the scheduler. These may be LUs associated with that scheduler, or LUs not associated

with any scheduler (NOSCHED LUs). Change the TP to use the ATBALC2 version of Allocate, and specify the name of an LU available to the scheduler on the Local_LU parameter.

- Ensure that the TP tries to allocate or initialize the conversation from an address space that is able to use the system base LU.

Source

APPC/MVS

Detecting Module: ATBVSAL, ATBVSIN

ATB80034I No system base LU defined to APPC/MVS.

Explanation

A TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service. The request did not specify a local LU name, so APPC/MVS had to select a local LU as the source of the conversation. The program is running in an address space that is not connected to a scheduler. No system base LU is defined, so APPC/MVS could not determine which LU to use as the source of the conversation.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate or Initialize_Conversation service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine if a NOSCHED LU is available.

At the request of the system programmer, enter a SET APPC=xx command to activate a newly defined LU in the APPC/MVS configuration.

System programmer response

At the request of the application programmer, enter an LUADD statement in an APPCPMxx parmlib member to define a NOSCHED LU or a NOSCHED base LU. Then ask the operator to enter a SET APPC=xx command to activate the newly defined LU in the APPC/MVS configuration.

Programmer response

Contact the operator to determine if a NOSCHED LU is available. If one is available, specify the name of the NOSCHED LU as the local LU on the call. If one is not available, do one of the following:

- Ask the system programmer to define a NOSCHED LU to the APPC/MVS configuration. Then, if a call to Allocate was originally interrupted, replace the call to Allocate with the ATBALC2 version of Allocate. On the call, specify the name of the NOSCHED LU as the local LU.
- Ask the system programmer to define a NOSCHED base LU to the APPC/MVS configuration.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSIN

ATB80036I **Buffer storage not available for
Receive processing.**

Explanation

A TP called an APPC/MVS conversation service. APPC/MVS could not obtain enough storage to store data to be sent or received in the conversation. This might be an intermittent error.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

System programmer response

Ensure that the amount of buffer storage specified on the BUFSTOR parameter in the start procedure for APPC/MVS (ATBINITM in SYS1.PROCLIB) is large enough to handle the data to be transmitted, and that the CONVBUFF parameter specifies a percentage of buffer storage that is large enough to handle the sending and receiving of data for the conversation. If you accepted the default value for BUFSTOR (which is approximately one third of free auxiliary storage) when you started APPC, ensure that the system has enough auxiliary storage to handle APPC processing. For more information about how to control the buffer storage limit, see *z/OS MVS Planning: APPC/MVS Management*. If the problem persists, contact the IBM Support Center.

Programmer response

Try to run the application again. If the error persists, contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBAMCU, ATBAMRC, ATBAMEL

ATB80037I **Value specified on
Conversation_type parameter is
not valid.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Set_Conversation_Type service to set the Conversation_type characteristic for a conversation. The Conversation_type parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Allocate service, see the description of the Allocate service in “Allocate” on page 125 for explanations of valid Conversation_type values. Enter a valid Conversation_type value on the call to the Allocate service.
- If the TP called the CPI-C Set_Conversation_Type service, see the description of the Set_Conversation_Type service in *CPI-C Reference* for a description of valid values for this parameter. Enter a valid Conversation_type value on the call.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSST

ATB80038I **Value specified on Sync_level
parameter is not valid.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Set_Sync_Level service to set the Sync_level characteristic for a conversation. The Sync_level parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Allocate service, see the description of the Allocate service in “Allocate” on page 125 for explanations of valid Sync_level values. Enter a valid Sync_level value on the call to the Allocate service.
- If the TP called the CPI-C Set_Sync_Level service, see the description of the Set_Sync_Level service in the *CPI-C Reference* for explanations of valid values for the sync_level parameter. Specify a valid value on the sync_level parameter.

Source

APPC/MVS

Detecting Module:
ATBVSSV, ATBVSAL

ATB80039I **Value specified on Security_type parameter is not valid.**

Explanation

A TP called the LU 6.2 Allocate service to allocate a conversation with another program. The Security_type parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Programmer response

See the description of the Allocate service in “Allocate” on page 125 for explanations of valid Security_type values. Enter a valid Security_type value on the call to the Allocate service.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80040I **Value specified on TP_name_length parameter is not valid.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Set_TP_Name service to set the TP_name characteristic for a conversation. The TP_name_length parameter specified a value that is not valid.

System action

One of the following:

- If the TP called the LU 6.2 Allocate service, the system returns a parameter_error (decimal 19) return code to the caller
- If the TP called the CPI-C Set_TP_Name service, the system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

One of the following:

- If the TP called the LU 6.2 Allocate service, specify a value between 0 and 64 on the TP_name_length parameter
- If the TP called the CPI-C Set_TP_Name service, specify a value between 1 and 64 on the TP_name_length parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSSV

ATB80041I **TP name length is zero, but no symbolic destination name is specified.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The TP_name_length parameter specified a value of zero, indicating that the system should use the

symbolic destination name to determine the name of the partner TP. The call to the Allocate service specified a blank symbolic destination name on the Sym_dest_name parameter, which is not allowed when the TP_name_length parameter specifies a value of zero.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Programmer response

When calling the Allocate service, do one of the following:

- Specify a valid symbolic destination name on the Sym_dest_name parameter
- Enter the name of the partner TP on the TP_name parameter, and enter the length of the TP name on the TP_name_length parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80042I **Caller is not authorized to specify User_token or TP_ID parameters.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The caller is not running in supervisor state or PSW key 0-7. The caller specified one or both of the following:

- A field on the User_token parameter whose first byte *does not* contain a hexadecimal zero (X'00')
- A field on the TP_ID parameter that *does not* contain all binary zeros.

To specify a User_token or TP_ID, the TP must be running in supervisor state or PSW key 0-7.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Programmer response

Do one of the following:

- Issue the MODESET assembler macro to enter supervisor state, or issue the SPKA assembler instruction to change to PSW key 0-7, before the TP calls the Allocate service
- Enter a User_token whose first byte contains a hexadecimal zero (X'00') and a TP_ID that contains all binary zeros (unauthorized programs must specify these values when calling the Allocate service).

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80043I **Calling program did not specify both a user ID and a password and/or surrogate authorization check failed.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The Security_type parameter specified a value of Security_pgm. This Security_type requires one of the following:

- Both the User_ID and Password parameters contain values, or
- The User_ID parameter contains a valid value and the TP calling the Allocate service has been granted surrogate user authority for this User_ID. This implies that this User_ID is a valid MVS user ID on the allocating system. For example, the user ID can be no more than eight characters long.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Programmer response

Specify valid values on the User_ID and Password parameters or specify a valid User_ID value and obtain surrogate user authorization for the User_ID. For more information, see [z/OS MVS Planning: APPC/MVS Management](#).

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80044I **Value *symdestname* specified on Sym_dest_name parameter is not defined in active side information data set.**

Explanation

A TP tried to allocate or initialize a conversation with another program. The Sym_dest_name parameter specified a value that is not defined in the active side information data set.

In the message text:

symdestname

The symbolic destination name specified on the Sym_dest_name parameter that could not be found in the active side information data set.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate or Initialize_Conversation service.

System programmer response

At the request of the application programmer, add the symbolic destination name to the side information data set (using the DESTNAME parameter).

Programmer response

Do *one* of the following:

- Validate that the correct symbolic destination name is specified on the Sym_dest_name parameter. If the name is correct, ask the system programmer to add the name to the side information data set.
- Specify eight blanks on the Sym_dest_name parameter, and specify valid values on the TP_name and Partner_LU_name parameters on the call to Allocate.

Source

APPC/MVS

Detecting Module:

ATBSD1G

ATB80045I **Mode name SNASVCMG is not valid.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The Mode_name

parameter specified the value SNASVCMG, which is a reserved logon mode that VTAM uses to exchange information with other LUs. You cannot specify SNASVCMG as a logon mode on calls to the Allocate service.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

System programmer response

At the request of the application programmer, provide a valid mode name (one that is defined in the logon mode table, a compiled version of which exists in SYS1.VTAMLIB). If necessary, update the mode name in the side information data set.

Programmer response

If necessary, ask the system programmer to provide a valid mode name. Then do one of the following:

- Specify a valid mode name on the Mode_name parameter
- Ask the system programmer to update the mode name for Sym_dest_name used in the side information data set.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSCA

ATB80046I **An allocate request for an SNA service TP name beginning with X'06' is allowed only for programs running in supervisor state or with PSW key 0-7.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The call specified an SNA service TP name that begins with X'06', which is not valid unless the caller is running in supervisor state or with PSW key 0-7.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Programmer response

Specify a valid SNA service TP name for the caller's environment, or switch the program to run in supervisor state or with PSW key 0-7.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSCA

ATB80047I **Value specified on User_token parameter is not valid.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS could not process the User_token specified on the call.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service. The security product issues abend X'9C7' to indicate that the request specified an incorrect User_token. The security product writes a logrec data set record that describes the error.

System programmer response

See the documentation for the security product to determine the response to the situation described in the logrec data set record.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80048I **Value specified on Local_LU_name parameter is not valid.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS could not process the Local_LU_name specified on the call. The specified Local LU is associated with a scheduler

other than the scheduler that is connected to the address space in which the TP is running.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Programmer response

Specify one of the following on the Local_LU_name parameter:

- Eight blanks, which will cause APPC/MVS to default to use the base LU for the scheduler
- The name of an LU owned by the scheduler associated with address space
- The name of a NOSCHED LU.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80049I **Value specified on Local_LU_name parameter is not the name of the system base LU or the name of a NOSCHED LU.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS could not process the local LU name specified on the call because all of the following conditions exist:

- The address space from which the allocate request was issued is not connected to a scheduler
- The specified local LU is not the system base LU
- The specified local LU is not a NOSCHED LU.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to display the LUs that are currently defined to the APPC/MVS configuration. Provide the application programmer with a valid LU name.

At the request of the system programmer, enter a SET APPC=xx command to add the newly defined LU to the APPC/MVS configuration.

System programmer response

If necessary, define a NOSCHED LU or a system base LU in the APPCPMxx parmlib member. Ask the operator to enter a SET APPC command to add the LU to the APPC/MVS configuration.

Programmer response

Do one of the following:

- Ask the operator to provide the name of the system base LU or the name of a NOSCHED LU; specify the name of the LU on the Local_LU_name parameter
- Specify eight blanks on the Local_LU name parameter, which causes APPC/MVS to default to use the system base LU.

Specify one of the listed values on the Local_LU name parameter.

Source

APPC/MVS

Detecting Module: ATBVSAL	
--	--

ATB80050I	Allocate request specified system base LU name on Local_LU_name parameter. Alternate scheduler cannot use that LU name.
------------------	--

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS could not process the local LU name specified on the call. The TP is running in an address space that is associated with an alternate scheduler. The specified local LU name is the system base LU name, but the LU is associated with the APPC/MVS transaction scheduler. TPs running under an alternate scheduler cannot specify the LU as the local LU on an allocate request.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to display the LUs that are currently defined to the APPC/MVS configuration. Provide the application programmer with a valid LU name.

At the request of the system programmer, enter a SET APPC=xx command to add a newly defined LU to the APPC/MVS configuration.

System programmer response

If necessary, define a NOSCHED LU or an LU associated with the alternate scheduler in the APPCPMxx parmlib member. Ask the operator to enter a SET APPC=xx command to add the LU to the APPC/MVS configuration.

Programmer response

Ask the operator to provide one of the following:

- The name of the LU associated with the alternate transaction scheduler
- The name of a NOSCHED LU.

Specify the provided LU name on the Local_LU name parameter.

Source

APPC/MVS

Detecting Module: ATBVSAL	
--	--

ATB80051I	Allocate request for SNA service TP is not valid when Conversation_type is mapped.
------------------	---

Explanation

A TP called the Allocate service to allocate a conversation with a SNA service TP. The call specified a Conversation_type of 1 (Mapped_conversation), which is not valid when allocating a SNA service TP.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Programmer response

Specify a conversation_type of 0 (Basic_conversation) on the Conversation_type parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSCA

ATB80052I LU specified on Local_LU_name parameter is not defined to APPC/MVS.

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS cannot process the LU name specified on the Local_LU_name parameter. The LU is not defined to APPC/MVS.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to display the LUs that are currently defined to the APPC/MVS configuration. Provide the application programmer with a valid LU name.

Programmer response

Contact the operator to determine which LUs are currently defined to the APPC/MVS configuration. Enter one of the following on the Local_LU_name parameter:

- A valid LU name
- Eight blanks, which will cause APPC/MVS to default to use the base LU for the scheduler.

If necessary, see the description of the Allocate service in [“Allocate” on page 125](#) for the types of local LUs for which an address space can allocate.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80053I Value specified on TP_ID parameter is not valid.

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS cannot find the TP associated with the TP_ID specified on the call. The TP does not exist in the caller's address space (it might exist in another address space).

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Programmer response

Specify a valid TP_ID on the call to the Allocate service.

Source

APPC/MVS

Detecting Module:
ATBVSAL

ATB80054I Value specified on Timeout_Value_Minutes parameter is not valid.

Explanation

A TP called the LU 6.2 Allocate or Set_TimeOut_Value service. The Timeout_Value_Minutes parameter specified a value that is not valid.

System action

The system returns a Program_Parameter_Check (decimal 24) return code to the caller of the specified conversation service.

Programmer response

See [“Allocate” on page 125](#) or [“Set_TimeOut_Value” on page 254](#) for an explanation of the Timeout_Value_Minutes parameter. Specify a valid value on the Timeout_Value_Minutes parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSST

ATB80055I **APPC/MVS could not create required internal data structures.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS tried to build the data structures that are necessary to maintain a conversation. APPC/MVS could not create those data structures at the time it issued this message.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate service.

System programmer response

Contact the IBM Support Center.

Programmer response

Call the Allocate service again. If the problem persists, contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBVSAL

ATB80058I **An abend occurred in APPC/MVS.**

Explanation

A TP called an APPC/MVS conversation service. An abend occurred while APPC/MVS was processing the conversation service.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the service. The system requests an SVC dump.

System programmer response

Contact the IBM Support Center. Provide the SVC dump.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

All

ATB80059I **Value *plu_name* specified on Partner_LU_name parameter is not valid.**

Explanation

A TP called the LU 6.2 or CPI-C Allocate service to allocate a conversation with another program. The request specified a partner LU name that is not valid. The following are examples of possible reasons why the partner LU name is not valid:

- It contained one or more null characters; the partner LU name must contain characters from the type A character set
- It contained one or more imbedded blanks (trailing blanks are allowed for the LU 6.2 Allocate service)
- It contained one or more imbedded or trailing blanks (trailing blanks are not allowed for CPI-C Allocate service).

Note: The trailing blank will not appear in the message text.

- It contained more than eight characters and was *not* network-qualified; a partner LU name that is not network-qualified cannot contain more than eight characters.

In the message text:

plu_name

The partner LU name specified on the Partner_LU_name parameter.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Allocate service, see the description of the Allocate service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125](#) for explanations of valid values for the Partner_LU_name parameter. Specify a valid value on the Partner_LU_name parameter.
- If the TP called the CPI-C Allocate service, see the description of the Set_Partner_LU_Name service in

CPI-C Reference for explanations of valid values for the partner_LU_name parameter.

Source

APPC/MVS

Detecting Module:

ATBAMAL

ATB80060I **Value specified on the Return_control parameter is not valid.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Set_Return_Control service to set the Return_control characteristic for a conversation. The Return_control parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Allocate service, see the description of the Allocate service in [“Allocate” on page 125](#) for explanations of valid Return_control values. Specify a valid value on the Return_control parameter.
- If the TP called the CPI-C Set_Return_Control service, see the description of the CPI-C Set_Return_Control service in *CPI-C Reference* for explanations of valid values for the return_control parameter. Specify a valid value on the return_control parameter.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSSV

ATB80061I **Session for mode name mode_name is not available. Partner LU: plu_name, local LU: loclu_name.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The caller specified a value of Immediate (1) on the Return_control parameter. No session is currently available for the mode name, so APPC/MVS cannot establish the conversation.

In the message text:

mode_name

The mode name specified in the side information or on the Mode_name parameter on the call to Allocate.

plu_name

The partner LU name specified on the Partner_LU_name parameter.

locclu_name

The name of the local LU for which APPC/MVS cannot establish a session.

System action

The system returns an unsuccessful (decimal 28) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU,[LIST|ALL] command to determine if a session is available for the specified local and partner LUs.

Programmer response

Do one of the following:

- Specify a value of When_session_allocated (0) on the Return_control parameter
- Call the Get_TP_Properties service to return the local and partner LU names for which a session is not available. Ask the operator to provide a mode name that has available sessions for the specified LUs. Specify that mode name on the Mode_name parameter.

Source

APPC/MVS

Detecting Module:

ATBLUMB

ATB80062I **Local LU LU_name is terminating. The system cannot establish a session.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS cannot establish a session because the local LU is terminating.

In the message text:

LU_name

The name of the local LU that is terminating.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU,SUMMARY command to verify that the LU is not active. If the LU is not active, enter a SET APPC=xx command to activate the LU. On the command, specify the APPCPMxx parmlib member in which the LU is added.

Programmer response

Verify that a name is specified on the Local_LU_name parameter and that the name is correct. If the name is correct, contact the operator to determine if the status of the LU is expected.

Source

APPC/MVS

Detecting Module:
ATBLUMB

ATB80063I **No default mode name is available to establish a session.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The request did not specify a mode name in the side information or on the Mode_name parameter, so APPC/MVS tried to find a mode name that was in effect for the local and partner LUs. APPC/MVS could not find a mode name, which is necessary to define network properties for the session.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

System programmer response

At the request of the application programmer, provide a valid mode name (one that is defined in the logon mode table, a compiled version of which exists in SYS1.VTAMLIB).

Programmer response

Ask the system programmer to provide a valid mode name. Specify a valid mode name on the Mode_name parameter.

Source

APPC/MVS

Detecting Module:
ATBLUMB

ATB80064I **Local LU LU_name is not active. APPC/MVS cannot establish a session.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS cannot establish a conversation because the local LU specified on the Local_LU_name parameter is not in "active" or "outbound only" state.

In the message text:

LU_name

The name of the local LU that is not active.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

System programmer response

At the request of the application programmer, ensure that the local LU is defined correctly in the VTAM application (APPL) statement in SYS1.VTAMLST.

Programmer response

Verify that the name specified on the Local_LU_name parameter is correct. If the name is correct, contact the system programmer to determine why the LU is not in "active" or "outbound only" state.

Source

APPC/MVS

Detecting Module:**ATBLUMB**

ATB80065I **Partner LU *plu_name* is not active.
APPC/MVS cannot establish a
session.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. APPC/MVS cannot establish the conversation. The specified partner LU is defined to APPC/MVS on this system, but the status of the LU is not "active" or "outbound only".

In the message text:

plu_name

The partner LU name specified in the side information or on the Partner_LU_name parameter.

System action

The system returns a parameter_error (decimal 19) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine the status of the partner LU. If the status of the partner LU is not "active" or "outbound only", enter a SET APPC=xx command to activate the LU.

Programmer response

Verify that the name specified on the Partner_LU_name parameter is correct. If the name is correct, contact the operator to determine why the status of the LU is not "active" or "outbound only".

Source

APPC/MVS

Detecting Module:**ATBLUMB**

ATB80066I **The Local LU unregistered as a
resource manager causing the
conversation to terminate.**

Explanation

A TP called a conversation service for a conversation with a synchronization level of syncpt, but the request could not be processed because the local LU became

unregistered as a resource manager, which resulted in the conversation being terminated abnormally. The local LU may have unregistered as a resource manager for one of the following reasons:

- An LUDEL was issued to quiesce work for the local LU and delete the LU from the APPC configuration.
- The transaction scheduler for the local lu unidentified itself from APPC/MVS.
- The syncpoint manager (RRS) became unavailable.
- A VTAM VARY command was issued to stop work for the local LU immediately.
- VTAM became unavailable.
- An internal APPC error caused the local LU to become unregistered as a resource manager.

System action

The system returns a product_specific_error (decimal 20) return code to the caller. No further conversation services will be successful for the conversation.

Operator response

At the request of the application programmer:

- Enter a DISPLAY APPC,LU command to display the LUs that are currently defined to the APPC/MVS configuration. Verify that the Local LU is defined to the APPC/MVS configuration, is registered as a resource manager and is capable of processing conversations with a synchronization level of syncpt.
- Verify that RRS is available.
- Verify that VTAM is available and the local LU is active in the VTAM configuration.

System programmer response

If the operator determines that VTAM and RRS are available and the local LU is defined and active in the APPC/MVS configuration, but the local LU is still not capable of processing conversations with a synchronization level of syncpt, search the problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the logrec data set error records and any SVC dumps taken by APPC/MVS.

Programmer response

Contact the operator to determine why the local LU may have unregistered as a resource manager and have the Operator verify that the Local LU is currently registered as a resource manager before attempting to the allocate another syncpt conversation.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSCA, ATBVSRB

ATB80067I The Local LU unregistered as a resource manager causing the allocate request to fail.

Explanation

A TP called the LU 6.2 Allocate service or the CPI-C Allocate service to allocate a conversation with a synchronization level of syncpt, but the request could not be processed because the local LU became unregistered as a resource manager, which resulted in the allocate request failing. The local LU may have unregistered as a resource manager for one of the following reasons:

- An LUDEL was issued to quiesce work for the local LU and delete the LU from the APPC configuration.
- The transaction scheduler for the local lu unidentified itself from APPC/MVS.
- The syncpoint manager (RRS) became unavailable.
- A VTAM VARY command was issued to stop work for the local LU immediately.
- VTAM became unavailable.
- An internal APPC error caused the local LU to become unregistered as a resource manager.

System action

The system returns a product_specific_error (decimal 20) return code to the caller. No further conversation services will be successful for the conversation.

Operator response

At the request of the application programmer:

- Enter a DISPLAY APPC,LU command to display the LUs that are currently defined to the APPC/MVS configuration. Verify that the Local LU is defined to the APPC/MVS configuration, is registered as a resource manager and is capable of processing conversations with a synchronization level of syncpt.
- Verify that RRS is available.
- Verify that VTAM is available and the local LU is active in the VTAM configuration.

System programmer response

If the operator determines that VTAM and RRS are available and the local LU is defined and active in the APPC/MVS configuration, but the local LU is still not capable of processing conversations with a synchronization level of syncpt, search the problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the logrec data set error records and any SVC dumps taken by APPC/MVS.

Programmer response

Contact the operator to determine why the local LU may have unregistered as a resource manager and have the Operator verify that the Local LU is currently registered as a resource manager before attempting to the allocate another syncpt conversation.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSCA, ATBVSRB

ATB80068I Value specified on Notify_type parameter is not valid.

Explanation

A TP called an APPC/MVS conversation service. The value specified on the Notify_type parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the specified service.

Programmer response

See the description of the conversation service in Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125 for explanations of valid Notify_type values. Specify a valid Notify_type value on the call to the service.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSCF, ATBVSCD, ATBVSE, ATBVSPT, ATBVSRT, ATBVSSR, ATBVSFL, ATBVSRC, ATBVSSD

ATB80069I Confirm processing not allowed when Sync_level is None.

Explanation

A TP called an APPC/MVS conversation service to send a confirmation request to a partner program on a conversation that was allocated with a Sync_level of None. A TP cannot send a confirmation request when a Sync_level of None is specified.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called LU 6.2 Allocate, ensure that the call to Allocate specifies a Sync_level of Confirm
- If the TP uses the CPI-C Initialize_Conversation and Allocate calls, specify a Sync_level of Confirm using the Set_Sync_Level (CMSSL) service
- Change the TP so it does not request confirm processing for the conversation.

Source

APPC/MVS

Detecting Module:

ATBVSCF, ATBVSSD, ATBVSDE, ATBVSPT

ATB80070I Value specified on Deallocate_type parameter is not valid.

Explanation

A TP called the Deallocate or Set_Deallocate_Type service to deallocate a conversation. The value specified on the Deallocate_type parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called LU 6.2 Deallocate, see the description of the Deallocate service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,”](#) on page 125 for explanations of

the valid Deallocate_type values. Enter a valid Deallocate_type on the call to the Deallocate service.

- If the TP called the CPI-C Set_Deallocate_Type service, see the description of the CPI-C Set_Deallocate_Type service in the *CPI-C Reference* for explanations of valid values for the deallocate_type parameter. Specify a valid value on the deallocate_type parameter.

Source

APPC/MVS

Detecting Module:

ATBVSDE, ATBVSST

ATB80071I Scheduler extract exit is not specified.

Explanation

A TP called one of the following services:

- LU 6.2 Allocate
- CPI-C Initialize_Conversation
- LU 6.2 Get_Conversation
- CPI-C Accept_Conversation.

The TP was running in an address space that had more than one active TP. APPC/MVS could not associate the request with a TP because the transaction scheduler for the address space did not specify an extract exit.

System action

If the TP called the LU 6.2 Allocate or CPI-C Initialize_Conversation service, the system returns a product_specific_error (decimal 20) return code to the caller. If the TP called the LU 6.2 Get_Conversation or CPI-C Accept_Conversation service, the system returns a program_state_check (decimal 25) return code to the caller.

System programmer response

Contact the owner of the scheduler product. Ask the owner of the scheduler to ensure that the scheduler product specifies an extract exit. See *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for information about how to establish an extract exit.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVSGC, ATBVSIN

ATB80073I TP cannot call the *service_name* service while in *current_state* state.

Explanation

A TP called an APPC/MVS conversation service. The call is not allowed in the conversation state for the conversation specified on the service call.

In the message text:

service_name

The name of the APPC/MVS TP conversation service that the TP tried to call.

current_state

The conversation state that the TP was in when it tried to call the service specified in the message text. One of the following values can appear in this field:

- Initialize
- Send
- Receive
- Send Pending
- Confirm
- Confirm Send
- Confirm Deallocate

System action

The system returns a program_state_check (decimal 25) return code to the caller of the conversation service.

Programmer response

See the APPC/MVS conversation state table in [Appendix C, “APPC/MVS Conversation State Table,” on page 399](#) for information about when you can call the APPC/MVS service specified in the message text. Change the TP so it calls the service while the conversation is in a valid state for that service.

Source

APPC/MVS

Detecting Module:

ATBVSFS

ATB80074I Call to previous service did not finish sending logical record. TP must specify more data to send.

Explanation

A TP tried to call an APPC/MVS conversation service. The conversation was in "send" state when the TP tried to call the service. A call to a previous conversation service did not send a complete logical record. For a basic conversation in "send" state, the TP cannot change the state of the conversation until a previous conversation service sends a complete logical record. Also, the TP cannot call the Confirm service until the complete logical record is sent.

System action

The system returns a program_state_check (decimal 25) return code to the caller of the conversation service.

Programmer response

Change the TP so it does not call another service before APPC/MVS finishes sending a complete logical record.

Source

APPC/MVS

Detecting Module:

ATBVSCF, ATBVSPT, ATBVSDE, ATBVSRC, ATBMSLL

ATB80075I Partner ended session for LU=LOCAL conversation.

Explanation

A TP tried to call an APPC/MVS service for an LU=LOCAL conversation. One of the following occurred:

- The partner TP called the Deallocate service to deallocate the conversation abnormally
- The partner TP was cancelled while the system was processing the conversation service.

System action

The system returns a resource_failure_retry (decimal 27) return code to the caller of the conversation service.

Programmer response

Determine why the partner TP ended the conversation (or why the partner TP was cancelled).

Source

APPC/MVS

Detecting Module:
ATBAMLS, ATBAMLR

ATB80076I **No data immediately available to receive for Receive_Immediate service.**

Explanation

A TP called the LU 6.2 Receive_Immediate or CPI-C Receive conversation service to receive information that is available to a conversation. No data was available to receive.

System action

The system returns an unsuccessful (decimal 28) return code to the caller of the Receive_Immediate or CPI-C Receive service.

Programmer response

Change the TP so it calls the Receive_and_Wait or Post_on_Receipt service before it calls the Receive_Immediate service. See the descriptions of these services in Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125 for more information.

Source

APPC/MVS

Detecting Module:
ATBAMRC, ATBAMLR

ATB80077I **Value specified on Error_Direction parameter is not valid.**

Explanation

A TP called the LU 6.2 Send_Error or CPI-C Set_Error_Direction service to inform a partner program that the TP encountered an error. The value specified on the Error_Direction parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the conversation service.

Programmer response

See the description of the Send_Error service in Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125 for explanations of valid Error_Direction values. Specify a valid Error_Direction value on the call.

Source

APPC/MVS

Detecting Module:
ATBVSSR, ATBVSST

ATB80078I **No inbound conversations available, no active TPs in the address space.**

Explanation

A TP called the Accept_Conversation or Get_Conversation service to obtain the conversation ID for an inbound conversation. No inbound conversation exists because there is no active TP in the address space.

System action

The system returns a program_state_check (decimal 25) return code to the caller of the Accept_Conversation or Get_Conversation service.

System programmer response

From the transaction scheduler in use, call the Associate service to associate the TP with the address space in which it is running.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSGC

ATB80079I **No inbound conversation available for the TP to receive.**

Explanation

A TP called the Accept_Conversation or Get_Conversation service to obtain the conversation ID for an inbound conversation. No inbound conversation exists to be received.

System action

The system returns a program_state_check (decimal 25) return code to the caller of the Accept_Conversation or Get_Conversation service.

Programmer response

Determine if the conversation was deallocated before the TP called the Accept_Conversation or Get_Conversation service. If so, change the TP so it does not call the service after the conversation is deallocated. If not, ensure that the conversation is allocated before trying to accept it or return its conversation ID.

Source

APPC/MVS

Detecting Module:
ATBVSGC

ATB80082I **A TP called Accept_Conversation or Get_Conversation out of sequence.**

Explanation

A TP called the Accept_Conversation or Get_Conversation service to return the conversation ID that the TP will use to reference the conversation on which it was allocated. APPC/MVS could not process the request because the TP called another APPC/MVS service previously (after the conversation was allocated).

System action

The system returns a program_state_check (decimal 25) return code to the caller of the Accept_Conversation or Get_Conversation service.

Programmer response

Change the TP so it calls the Accept_Conversation or Get_Conversation service before it calls any

other APPC/MVS service (after the conversation is allocated).

Source

APPC/MVS

Detecting Module:
ATBVSGC

ATB80083I **Side information data set not defined.**

Explanation

A TP called the Allocate service to allocate a conversation with another program. The request specified a symbolic destination name on the Sym_dest_name parameter, but the side information data set was not defined to APPC/MVS.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Allocate service.

Operator response

At the request of the application programmer, enter a DISPLAY APPC,LU command to determine if a side information data set is defined to APPC/MVS. If not, ask the system programmer to define the data set.

At the request of the system programmer, enter a SET APPC=xx command to add the side information data set specified in the APPCPMxx parmlib member.

System programmer response

At the request of the operator, add a SIDEINFO statement that defines the side information data set to an APPCPMxx parmlib member. Then ask the operator to enter a SET APPC=xx command to add the side information data set specified in the APPCPMxx parmlib member.

Programmer response

Contact the operator to determine if a side information data set is defined to APPC/MVS.

Source

APPC/MVS

Detecting Module:

ATBSD1G

**valid when Fill characteristic is
CM_fill_buffer.**

ATB80084I **From VTAM macro APPCCMD:
General Return Code: *genrc*,
Recovery Action Return Code:
*recrc***

Explanation

A TP called an APPC/MVS conversation service that resulted in an invocation of the VTAM APPCCMD macro. The APPCCMD macro returned a non-zero return code, indicating that an error occurred.

In the message text:

genrc

The general return code from the VTAM macro APPCCMD.

recrc

The recovery action return code from the VTAM macro APPCCMD.

System action

The system returns a non-zero return code to the caller of the conversation service.

System programmer response

Search the system log and logrec data set for any messages indicating an error related to an APPC/MVS LU. Search the problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center with any diagnostic information that has been gathered.

Programmer response

See *z/OS Communications Server: SNA Programming* for explanations of the general return code and recovery action return code combination. The receipt of an unexpected general return code and recovery action return code from a VTAM APPCCMD usually means there is an internal APPC or VTAM error.

Source

APPC/MVS

Detecting Module:

**ATBAMAL, ATBAMCD, ATBAMCF, ATBAMDE,
ATBAMFL, ATBAMPT, ATBAMRE, ATBAMRT,
ATBAMSD, ATBAMSR, ATBAMTS, ATBLUVS**

ATB80085I **Conversation_type of
CM_mapped_conversation is not**

Explanation

A TP called the CPI-C Set_Conversation_Type service to set the conversation type characteristic for a conversation. The Conversation_type parameter specified a mapped conversation. Previously, the TP specified a value of CM_FILL_BUFFER on a call to the CPI-C Set_Fill service, which indicates that the TP is to receive data independent of its logical format. The Set_Conversation_Type service cannot specify a mapped conversation when a fill type of CM_FILL_BUFFER is specified.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Conversation_Type service.

Programmer response

Do one of the following:

- Set the conversation type to basic
- Call the Set_Fill service to specify a Fill of Fill_LL before calling Set_Conversation_Type to specify a Conversation_Type of mapped_conversation.

Source

APPC/MVS

Detecting Module:

ATBVSST

ATB80086I **Conversation_type of
CM_mapped_conversation is not
valid when log_data characteristic
is specified.**

Explanation

A TP called the CPI-C Set_Conversation_Type service to set the conversation type characteristic for a conversation. The Conversation_type parameter specified CM_MAPPED_CONVERSATION. Previously, the TP called the CPI-C Set_Log_Data service to set log data characteristics for the conversation. Because log data cannot be sent for a mapped conversation, APPC/MVS cannot process the request.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Conversation_Type service.

Programmer response

Do one of the following:

- Set the conversation type to basic (specify a value of CM_BASIC_CONVERSATION on a call to the Set_Conversation_Type service)
- Remove the call to Set_Log_Data from the TP.

Source

APPC/MVS

Detecting Module:
ATBVSST

ATB80087I **Fill of CM_fill_buffer is not valid when conversation_type characteristic is CM_mapped_conversation.**

Explanation

A TP called the CPI-C Set_Fill conversation service to set the fill characteristic for a conversation. The fill parameter specified that the TP is to receive data independent of its logical record format (a value of CM_FILL_BUFFER). Previously, the TP called the CPI-C Set_Conversation_Type service to set a mapped conversation type for the conversation (a value of CM_MAPPED_CONVERSATION). A TP cannot specify an independent logical format for a mapped conversation.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Fill service.

Programmer response

Do one of the following:

- Set the conversation type to basic (specify a value of CM_BASIC_CONVERSATION on the call to the Set_Conversation_Type service)
- Remove the call to Set_Fill from the TP.

Source

APPC/MVS

Detecting Module:
ATBVSST

ATB80088I **Value specified on Prepare_to_receive_type parameter is not valid.**

Explanation

A TP called the LU 6.2 Prepare_to_Receive or the CPI-C Set_Prepare_To_Receive_Type service to change a conversation from send to receive state. The value specified on the Prepare_to_receive_type parameter was not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Prepare_to_Receive service, see the description of the Prepare_to_Receive service in [Chapter 8, "APPC/MVS TP Conversation Callable Services," on page 125](#) for explanations of valid Prepare_to_receive_type values. Specify a valid value on the Prepare_to_receive_type parameter.
- If the TP called the CPI-C Set_Prepare_to_Receive_Type service, see the description of the CPI-C Set_Prepare_To_Receive_Type service in the *CPI-C Reference* for explanations of valid values for the prepare_to_receive_type parameter. Specify a valid value on the prepare_to_receive_type parameter.

Source

APPC/MVS

Detecting Module:
ATBVSPT, ATBVSSV

ATB80089I **Value specified on log_data_length parameter is not valid.**

Explanation

A TP called the CPI-C Set_Log_Data service to set the log data characteristic for a conversation. The log_data_length parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the CPI-C Set_Log_Data service.

Programmer response

See the description of the CPI-C Set_Log_Data service in the *CPI-C Reference* for explanations of valid values for the log_data_length parameter. Specify a valid value on the log_data_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSST

ATB80090I Set_Log_Data service not valid for a mapped conversation.

Explanation

A TP called the CPI-C Set_Log_Data service to set the log data characteristic for a conversation. The conversation is mapped. A TP cannot issue the Set_Log_Data service for a mapped conversation.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Log_Data service.

Programmer response

Change the TP so it does not call the Set_Log_Data service for a mapped conversation.

Source

APPC/MVS

Detecting Module:

ATBVSST

ATB80091I Value specified on Mode_name_length parameter is not valid.

Explanation

A TP called the CPI-C Set_Mode_Name service to set the mode for a conversation. The value specified on the mode_name_length parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Mode_Name service.

Programmer response

See the description of the CPI-C Set_Mode_Name service in the *CPI-C Reference* for explanations of valid values for the Mode_name_length parameter. Specify a valid value on the Mode_name_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSSV

ATB80092I Value specified on partner_lu_name_length parameter is not valid.

Explanation

A TP called the CPI-C Set_Partner_LU_name service to set the partner_LU_name characteristic for a conversation. The value specified on the partner_lu_name_length parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Partner_LU_Name service.

Programmer response

See the description of the CPI-C Set_Partner_LU_Name service in the *CPI-C Reference* for explanations of valid values for the partner_LU_name_length parameter. Specify a valid value on the partner_LU_name_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSSV

ATB80093I Deallocate_type of CM_deallocate_confirm is not valid when sync_level characteristic is CM_none.

Explanation

A TP called the CPI-C Set_Deallocate_Type service to set the deallocate type characteristic for a conversation. The deallocate_type parameter specified a value of CM_Deallocate_Confirm, which is not compatible with the established conversation sync_level of CM_none.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Deallocate_Type service.

Programmer response

Do *one* of the following:

- See the description of the CPI-C Set_Deallocate_Type service in the *CPI-C Reference* for explanations of valid values for the Deallocate_type parameter. Specify a value other than CM_Deallocate_Confirm on the deallocate_type parameter.
- See the description of the CPI-C Set_Sync_Level service in the *CPI-C Reference* for explanations of valid values for the sync_level parameter. Specify a value other than CM_none on the sync_level parameter.

Source

APPC/MVS

Detecting Module:
ATBVSST

ATB80094I Prepare_to_receive_type of CM_prep_to_receive_confirm is not valid when sync_level characteristic is CM_none.

Explanation

A TP called the CPI-C Set_Prepare_To_Receive_Type service to set the prepare to receive type characteristic for a conversation. The prepare_to_receive_type parameter specified a value of CM_Prep_to_Receive_Confirm, which is not compatible with the established conversation sync_level of CM_none.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Prepare_To_Receive_Type service.

Programmer response

Do *one* of the following:

- See the description of the CPI-C Set_Prepare_To_Receive_Type service in the *CPI-C Reference* for explanations of valid values for the prepare_to_receive_type parameter. Specify a value other than CM_Prep_to_Receive_Confirm on the prepare_to_receive_type parameter.
- See the description of the CPI-C Set_Sync_Level service in the *CPI-C Reference* for explanations of valid values for the sync_level parameter. Specify a value other than CM_none on the sync_level parameter.

Source

APPC/MVS

Detecting Module:
ATBVSSV

ATB80095I Value specified on Receive_type parameter is not valid.

Explanation

A TP called the CPI-C Set_Receive_Type service to set the receive_type characteristic for a conversation. The receive_type parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Receive_Type service.

Programmer response

See the description of the CPI-C Set_Receive_Type service in the *CPI-C Reference* for explanations of valid values for the receive_type parameter. Specify a valid value on the receive_type parameter.

Source

APPC/MVS

Detecting Module:
ATBVSSV

ATB80096I Deallocate_type of deallocate_confirm or deallocate_flush is not valid

when sync_level characteristic is Syncpoint.

Explanation

A TP called the LU 6.2 Deallocate service to deallocate a conversation with a synchronization level of syncpoint. The specified deallocate_type is not valid for deallocating a syncpt conversation. The valid deallocate types that can be specified for a syncpt conversation are deallocate_sync_level or deallocate_abend.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the LU 6.2 Deallocate service.

Programmer response

Specify a deallocate_type of deallocate_sync_level or deallocate_abend to deallocate the syncpoint conversation.

Source

APPC/MVS

Detecting Module:

ATBVSDE

ATB80097I Value specified on send_type parameter is not valid.

Explanation

A TP called the LU 6.2 Send_Data or CPI-C Set_Send_Type service to set the Send_type characteristic for a conversation. The Send_type parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller.

Programmer response

Do one of the following:

- If the TP called the LU 6.2 Send_Data service, see the description of the Send_Data service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125](#) for an explanation of the Send_type parameter. Specify a valid value on the Send_type parameter.

- If the TP called the CPI-C Set_Send_Type service, see the description of the CPI-C Set_Send_Type service in the *CPI-C Reference* for explanations of valid values for the Send_type parameter. Specify a valid value on the Send_type parameter.

Source

APPC/MVS

Detecting Module:

ATBVSSV, ATBVSSD

ATB80098I Send_type of CM_send_and_confirm is not valid when sync_level characteristic is CM_none.

Explanation

A TP called the CPI-C Set_Send_Type service to set the Send_type characteristic for a conversation. The send_type parameter specified a value of CM_Send_and_Confirm, which is not compatible with the established conversation sync_level of CM_none. A conversation must be able to perform confirmation processing to accept the CM_Send_and_Confirm.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Send_Type service.

Programmer response

Do one of the following:

- If you want to send a request for confirmation, specify a sync_level of CM_CONFIRM on a call to the Set_Sync_Level service. Then call the Set_Send_Type service again.
- If you do not want to send a request for confirmation, see the description of the CPI-C Set_Send_Type service in the *CPI-C Reference* for explanations of other of valid values for the send_type parameter.

Source

APPC/MVS

Detecting Module:

ATBVSSV

ATB80099I **The APPCCMD command detected that the ACB is no longer active for the local LU.**

Explanation

A TP called an APPC/MVS conversation service that resulted in an invocation of the VTAM APPCCMD macro. The APPCCMD macro returned a non-zero return code, indicating that an error occurred because the ACB for the local LU is closed.

System action

The system records the return codes from APPCCMD in a logrec record, and returns a non-zero return code to the caller of the conversation service.

Programmer response

See *z/OS Communications Server: SNA Programmer's LU 6.2 Guide* for explanations of the return code from the APPCCMD macro. Check the Error_Log_Information_Length parameter on Error_Extract to see if the partner system returned log data. If log data is available, see the description of the Error_Extract service “Error_Extract” on page 158 for information about how to use the log data to diagnose the error.

Source

APPC/MVS

Detecting Module:

ATBAMAL, ATBAMCD, ATBAMCF, ATBAMDE, ATBAMFL, ATBAMPT, ATBAMRE, ATBAMRT, ATBAMSD, ATBAMSR, ATBAMTS, ATBLUVS

ATB80100I **From VTAM macro APPCCMD:**
Primary error return code:
prim_code, **secondary error return code:** *sec_code*, **sense code:** *sense_code*.

Explanation

A TP called an APPC/MVS conversation service that resulted in an invocation of the VTAM APPCCMD macro. The APPCCMD macro returned a non-zero return code, indicating that an error occurred.

In the message text:

prim_code
The primary return code from the VTAM macro APPCCMD.

sec_code
The secondary return code from the VTAM macro APPCCMD.

sense_code
The sense code from the VTAM macro APPCCMD.

System action

The system returns a non-zero return code to the caller of the conversation service. The partner LU might return log data, which further describes the error.

System programmer response

Determine why the ACB is closed. It could be because VTAM is not active, the scheduler unidentified with the IMMEDIATE option or the LU was inactivated with a VTAM command. If so, activate the LU so that the ACB opens and the LU transitions to Active state.

See *z/OS Communications Server: SNA Programmer's LU 6.2 Guide* for explanations of the primary return code, secondary return code and the sense code displayed in the message text. Check the Error_Log_Information_Length parameter on Error_Extract to see if the partner system returned log data. If log data is available, see the description of the Error_Extract service “Error_Extract” on page 158 for information about how to use the log data to diagnose the error.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBAMAL, ATBAMCD, ATBAMCF, ATBAMDE, ATBAMFL, ATBAMPT, ATBAMRE, ATBAMRT, ATBAMSD, ATBAMSR, ATBAMTS, ATBLUVS

ATB80101I **LU=LOCAL conversation received sense code *sense_code* from partner TP.**

Explanation

A TP running on MVS called an APPC/MVS service for an LU=LOCAL conversation. The partner TP found an error and provided the sense code specified in the message text.

In the message text:

sense_code

The sense code provided by the partner TP.

System action

The system returns a non-zero return code to the caller of the conversation service. The partner TP might return log data, which further describes the error.

Programmer response

See [z/OS Communications Server: SNA Programmer's LU 6.2 Guide](#) for an explanation of the sense code displayed in the message text. Check the Error_Log_Information_Length parameter on Error_Extract to see if the partner system returned log data.

Source

APPC/MVS

Detecting Module:
ATBAMLR, ATBAMLS

ATB80102I **Sync_level of CM_none is not valid when send_type, prepare_to_receive_type or deallocate_type characteristic indicates confirm processing.**

Explanation

A TP called the CPI-C Set_Sync_Level service to set the synchronization level characteristic for a conversation. The value specified on the sync_level parameter is 0 (CM_none). A previous call to the Set_Send_Type, Set_Prepare_to_Receive_Type, or Set_Deallocate_Type service requested confirmation processing. A sync_level value of CM_none is not valid when confirmation processing is requested.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Sync_Level service.

Programmer response

Do one of the following:

- See the description of the CPI-C Set_Sync_Level service in the *CPI-C Reference* for explanations of valid values for the sync_level parameter. Specify a value other than CM_none (0) on the sync_level parameter.

- See the description of the CPI-C Set_Send_Type, Set_Prepare_to_Receive_Type, or Set_Deallocate_Type service in the *CPI-C Reference* for explanations of valid values for the parameter that specifies confirmation processing. Specify a value that does not request confirmation processing on that parameter.

Source

APPC/MVS

Detecting Module:
ATBVSSV

ATB80103I **Sync_level of CM_syncpt is not valid when deallocate_type characteristic is either deallocate_flush or deallocate_confirm.**

Explanation

A TP called either:

- The CPI-C Set_Sync_Level service to set the synchronization level characteristic for a conversation to sync_level_syncpt, or
- The CPI-C Set_Deallocate_Type service to set the deallocate type characteristic for a conversation to either deallocate_flush or deallocate_confirm, after a previous call to the Set_Sync_Level service set the synchronization level characteristic for a conversation to sync_level_syncpt.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Set_Sync_Level or Set_Deallocate_Type service.

Programmer response

Do one of the following:

- See the description of the Set_Sync_Level service in *CPI-C Reference* for explanations of valid values for the sync_level parameter when the Deallocate_Type characteristic for the conversation is set to either deallocate_flush or deallocate_confirm.
- See the description of the Set_Deallocate_Type service in *CPI-C Reference* for explanations of valid values for the deallocate_type parameter when the synchronization level characteristic of the conversation is set to CM_Syncpt. The only valid value for deallocate_type is cm_deallocate_sync_level when the synchronization

level characteristic of the conversation is set to CM_Syncpt.

Source

APPC/MVS

Detecting Module:
ATBVSST, ATBVSSV

ATB80104I **Value specified on Locks parameter is not valid.**

Explanation

A TP called the Prepare_to_Receive service to change a conversation from send to receive state. The value specified on the Locks parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Prepare_to_Receive service.

Programmer response

See the description of the Prepare_to_Receive service for an explanation of the Locks parameter. Specify a valid value on the Locks parameter.

Source

APPC/MVS

Detecting Module:
ATBVSPT

ATB80105I **Value specified on Fill parameter is not valid.**

Explanation

A TP called the LU 6.2 Receive_Immediate, LU 6.2 Receive_and_Wait, or CPI-C Set_Fill service. The value specified on the Fill parameter is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the specified conversation service.

Programmer response

See the description of the APPC/MVS conversation service in this document (if the TP called an LU 6.2

service) or the *CPI-C Reference* (if the TP called a CPI-C service) for explanations of valid values for the Fill parameter. Specify a valid value on the Fill parameter.

Source

APPC/MVS

Detecting Module:
ATBVSRC, ATBVSSST

ATB80106I **Value specified on Access_token parameter is not valid.**

Explanation

A TP called the Send_Data, Receive_and_Wait, or Receive_Immediate conversation service. The Access_token parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the specified conversation service.

Programmer response

See the description of the Send_Data, Receive_and_Wait, or Receive_Immediate service in Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125 for an explanation of the Access_token parameter. Specify a valid value on the Access_token parameter.

Source

APPC/MVS

Detecting Module:
ATBVSRC, ATBVSSD

ATB80107I **Value specified on Requested_length parameter is not valid.**

Explanation

A TP called the CPI-C Receive service to receive information from a conversation. The Requested_length parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the CPI-C Receive service.

Programmer response

See the description of the CPI-C Receive service in the *CPI-C Reference* for an explanation of the Requested_length parameter. Specify a valid value on the Requested_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSRC

ATB80108I Value specified on Receive_length parameter is not valid.

Explanation

A TP called the LU 6.2 Receive_Immediate or Receive_and_Wait service to receive information from a conversation. The Receive_length parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the specified conversation service.

Programmer response

See the description of the Receive_Immediate or Receive_and_Wait service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125](#) for an explanation of the Receive_length parameter. Specify a valid value on the Receive_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSRC

ATB80109I Value specified on Send_length parameter is not valid.

Explanation

A TP called the Send_Data service to send data to a partner program. The Send_length parameter specified a value that is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Send service.

Programmer response

Do one of the following:

- If the TP called LU 6.2 Send_Data, see the description of the Send_Data service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125](#) for explanations of the valid Send_length values. Enter a valid Send_length on the call to the Send_Data service.
- If the TP called the CPI-C Send_Data service, see the description of the CPI-C Send_Data service in the *CPI-C Reference* for explanations of valid values for the Send_length parameter. Specify a valid value on the Send_length parameter.

Source

APPC/MVS

Detecting Module:

ATBVSSD

ATB80110I Value specified on Buffer parameter is not valid for a basic conversation.

Explanation

A TP called the Send_Data service for a basic conversation. The buffer parameter contains a logical record for which the logical record length field is not valid.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Send service.

Programmer response

See the description of the Send_Data service in [Chapter 8, “APPC/MVS TP Conversation Callable Services,” on page 125](#) for explanations of valid values

for the Buffer parameter. Specify a valid value on the Buffer parameter.

Source

APPC/MVS

Detecting Module:

ATBMSLL

ATB80111I Post_On_Receipt service not valid for a mapped conversation.

Explanation

A TP called the Post_On_Receipt service to request notification when data or status is received for a specified conversation. The TP called Post_On_Receipt for a mapped conversation. APPC/MVS does not support Post_On_Receipt for mapped conversations.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of the Post_On_Receipt service.

Programmer response

Do one of the following:

- Allocate the conversation with a basic conversation type
- Remove the call to Post_On_Receipt from the TP.

Source

APPC/MVS

Detecting Module:

ATBVSPR

ATB80112I Protocol Violation: APPC/MVS received deallocation status on a conversation with Sync_Level of Syncpt, but not during a two-phase commit exchange.

Explanation

APPC/MVS received a deallocation status on a conversation with synchronization level of syncpt outside the scope of a two-phase commit exchange, which is a violation of the Syncpoint programming architecture.

System action

The system returns a resource_failure_no_retry_bo (decimal 133) return code to the caller of the conversation service. The current unit of recovery has been put into backout required state. Local protected resources should be backed out. APPC/MVS terminates the session with the partner program that sent the deallocation status.

Programmer response

Correct the application program to deallocate syncpoint conversations either normally as part of the two phase commit exchange or abnormally by deallocating the syncpoint conversation with a deallocate type of deallocate_abend_*.

Source

APPC/MVS

Detecting Module:

ATBAMRE

ATB80114I Protocol violation: A conversation with a Sync_Level of None was established, but APPC/MVS received confirm status.

Explanation

A TP tried to call an APPC/MVS conversation service. The conversation for which the service was called has a sync_level characteristic of None, which indicates that the TPs using this conversation will not perform confirmation processing. APPC/MVS received a request for confirmation on the conversation from the Partner TP.

System action

The system returns a resource_failure_no_retry (decimal 26) return code to the caller of the conversation service.

Programmer response

Establish the conversation with a sync_level of confirm.

Source

APPC/MVS

Detecting Module:

ATBAMRE

ATB80115I **Protocol violation: Partner system specified an incorrect GDS variable format for error log data.**

Explanation

A TP called an APPC/MVS conversation service. The partner LU previously sent error log data on this conversation. The general data stream (GDS) variable that contains the error log data does not conform to the LU 6.2 architecture.

System action

The system returns a resource_failure_no_retry (decimal 26) return code to the program that tried to receive the log data.

System programmer response

See the description of error log variables in [*z/OS Communications Server: SNA Programmer's LU 6.2 Guide*](#) for the correct GDS variable format. Contact the service department for the system on which the partner TP is running. Ensure that the partner system is passing a GDS variable that conforms to the APPC/MVS architecture.

Programmer response

Contact the system programmer. Provide the name of the partner system on which the partner program is running.

Source

APPC/MVS

Detecting Module: ATBAMEL
--

ATB80116I **Protocol violation: APPC/MVS failed to receive expected error log data.**

Explanation

A TP called an APPC/MVS conversation service. The Partner LU indicated that error log data would be sent for the conversation, but no error log data is available to receive or APPC/MVS service failed while receiving the error log data.

System action

The system returns a resource_failure_no_retry (decimal 26) return code to the caller of the conversation service.

System programmer response

Verify that APPC/MVS did not abend while receiving the error log data. Contact the service department for the system on which the partner TP is running. Ask the service personnel to change the partner TP so it either sends log data or does not incorrectly indicate that log data is to be sent.

Programmer response

Contact the system programmer. Provide the name of the partner system on which the partner program is running.

Source

APPC/MVS

Detecting Module: ATBAMEL
--

ATB80117I **Protocol violation: Partner system did not send complete error log data in GDS variable.**

Explanation

A TP called an APPC/MVS conversation service. The Partner LU indicated that error log data would be sent for the conversation, but APPC/MVS did not receive the complete log data.

System action

The system returns a resource_failure_no_retry (decimal 26) return code to the caller of the conversation service.

System programmer response

Contact the service department for the system on which the partner TP is running. Ask the service personnel to change the partner TP so it sends complete log data in the GDS variable.

Programmer response

Contact the system programmer. Provide the name of the partner system on which the partner program is running.

Source

APPC/MVS

Detecting Module:

ATBAMEL

ATB80119I **An OpenEdition MVS exit failed when a TP tried to allocate a conversation with the OpenEdition MVS fork TP.**

Explanation

A TP tried to allocate a conversation with the z/OS UNIX System Services fork TP. The z/OS UNIX System Services allocate conversation exit could not process the request.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the Allocate service.

Programmer response

Do not attempt to allocate the z/OS UNIX System Services fork TP directly. Use the z/OS UNIX System Services fork service to allocate the conversation with the z/OS UNIX System Services fork TP.

Source

APPC/MVS

Detecting Module:

ATBAMAL

ATB80120I **The conversation is not owned by the home address space of the caller of the service.**

Explanation

APPC/MVS considers the scope of a TP to be the home address space. Access to a conversation is limited to programs whose home address space is the same as the home address space of the TP that:

- Allocated the conversation
- Accepted the conversation
- Received the conversation via the Receive_Allocate service for APPC/MVS Server applications

Ownership of a TP and its conversations may also be reassigned to an address space other than the original

owning address space by using the Associate service for Transaction Schedulers.

System action

The system returns a program_parameter_check (decimal 24) return code to the caller of requested service.

System programmer response

Programmer response

Resubmit the request from an address space whose home address space is the same as the current owning home address space for conversation.

Source

APPC/MVS

Detecting Module:

ATBVSRB

ATB80121I **The local LU does not support the synchronization level specified.**

Explanation

A TP called the LU 6.2 Allocate or CPI-C Set_Sync_Level service to set the Sync_level characteristic for a conversation to sync_level_syncpt. A Sync_level parameter value of sync_level_syncpt is not supported by the local LU.

System action

For the LU 6.2 Allocate service, the system returns a program_parameter_check (decimal 24) return code to the caller. For the CPI-C Set_Sync_Level service, the system returns a cm_parm_value_not_supported (decimal 49) return code to the caller.

System programmer response

At the request of the application programmer, identify the contents of the VTAM application (APPL) statement in SYS1.VTAMLST for the local LU. To support protected conversations (that is, conversations with a synchronization level of syncpt), the VTAM APPL statement for the local LU must contain the keywords and values SYNCLVL=SYNCPT and ATNLOSS=ALL.

Programmer response

Determine what level of synchronization is supported by the local LU. The defined synchronization level of the local LU is defined in the VTAM application (APPL) statement in SYS1.VTAMLST.

Source

APPC/MVS

Detecting Module:
ATBVSSV, ATBVSAL

**ATB80122I System cannot process a call.
A Syncpoint or Backout verb is running.**

Explanation

A TP called an APPC/MVS TP conversation service. The system could not process the request because APPC/MVS data structures for the conversation are in use; a Commit (that is, Syncpoint) or Backout request was issued for a unit of recovery that includes the conversation for which the conversation service was issued.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

System programmer response

Using the DISPLAY APPC,TP and DISPLAY APPC,UR commands, determine whether the syncpoint operation is still in progress, and whether resynchronization is required.

Programmer response

Your TP can periodically retry the call to the conversation service; the call should be successful once the syncpoint operation completes. If this in-use condition persists, notify the system programmer.

Source

APPC/MVS

Detecting Module:
ATBVSRB

**ATB80123I System cannot process a call.
The conversation is in backout required state.**

Explanation

A TP called an APPC/MVS TP conversation service for a syncpt conversation. The system could not process the request because a "backout required" condition exists for the unit of recovery that the syncpt conversation belongs to.

System action

The system returns a program_state_check (decimal 25) return code to the caller of the conversation service.

Programmer response

When a unit of recovery that includes an application programs syncpoint conversation is in backout required state the application program should issue a Backout call to backout local resource associated with the current unit of recovery.

Source

APPC/MVS

Detecting Module:
ATBVSRB

ATB80124I The local LU cannot process syncpt requests at the present time.

Explanation

A TP called either:

- The Allocate service to allocate a protected conversation (a conversation with a synchronization level of syncpt), or
- The CPI-C Set_Sync_Level service to set the Sync_level characteristic for a conversation to sync_level_syncpt.

The request failed because the local LU for the TP is not registered as a resource manager with the system syncpoint manager (RRS). The local LU is not registered for one of the following reasons:

- The system syncpoint manager is not active.
- An error that occurred during resource manager restart processing has prevented the local LU from registering as a resource manager with the system syncpoint manager.

System action

The called service returns a product_specific_error (decimal 20) return code to the caller. If the Allocate service was called, a conversation was not allocated. If the CPI-C Set_Sync_Level service was called, the Sync_level characteristic for the conversation was not set.

Operator response

At the request of the application programmer, determine if the system syncpoint manager is active, and notify the system programmer.

At the request of the system programmer, take the necessary action to activate the system syncpoint manager, if it is not available.

System programmer response

If the system syncpoint manager is not active, determine why it is not active and what steps must be taken to activate it.

If RRS is available, have the operator enter a DISPLAY APPC,LU command to determine the syncpt capability of the local LU. If the local LU is not capable of processing protected conversations, search the system log and logrec data set for any messages indicating an error that prevented the local LU from registering as a resource manager.

Source

APPC/MVS

Detecting Module: ATBVSSV, ATBVSAL

ATB80125I	Log name exchange failed during syncpt processing.
------------------	---

Explanation

A TP called the LU 6.2 Allocate service or CPI-C Allocate Service to allocate a conversation with a synchronization level of sync_level_syncpt. The request failed because an error occurred during the exchange log name transaction that is performed as part of the allocate request to the partner lu.

System action

The called service returns a product_specific_error (decimal 20) return code to the caller. Additional diagnostic messages accompany this message to identify the specific reason for the exchange log name failure.

Programmer response

See the explanation for the ATB80xxxI message that accompanies this message.

Source

APPC/MVS

Detecting Module: ATBPCRS
--

ATB80126I	Conversation was terminated during syncpt processing.
------------------	--

Explanation

A TP called a conversation service that cannot be processed because a conversation failure occurred during a previous syncpt or backout verb request. The return code returned on the call represents the reason for the conversation failure.

System action

The called service returns a return code that represents the reason for the conversation failure. The conversation has terminated and all resources associated with the conversation have been cleaned up.

Source

APPC/MVS

Detecting Module: ATBVSRB
--

ATB80127I	Conversation was terminated because APPC/MVS received unrecognized PS header.
------------------	--

Explanation

APPC/MVS received an unrecognized PS (Presentation Services) Header. A PS Header is sent by a partner lu as a request for the local application program to commit local protected resources by issuing a Syncpt verb.

System action

The system returns a resource_failure_no_retry_bo (decimal 133) return code to the caller of the conversation service. The current unit of recovery has been put into backout required state. Local protected resources should be backed out. APPC/MVS

terminates the session and conversation with the partner LU that sent the invalid PS Header.

Source

APPC/MVS

Detecting Module:

ATBAMRE

ATB80128I **APPC/MVS detected a protocol violation during an Exchange Log Name processing.**

Explanation

During exchange-log-name processing, an LU detected an error in the data sent by its partner LU. Message ATB80125I accompanies this message, and ATB206E is also issued to provide additional diagnostic information.

System action

The protected conversation allocate request was unsuccessful. Until the protocol violation being made by the partner LU is corrected, no protected conversations between the local and partner LU can be allocated.

Additional diagnostic information is written to the logrec data set to assist in diagnosing the problem.

The LU that made the protocol violation receives message ATB70051I as log data.

System programmer response

Contact the designated support group for your installation.

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB80129I **APPC/MVS detected a warm/cold log status mismatch during an Exchange Log Name processing.**

Explanation

While APPC/MVS was processing an Allocate call, exchange-log-name processing was required. During the exchange processing, an LU detected a warm/cold mismatch. Message ATB80125I accompanies this message, and ATB210E is also issued to provide additional diagnostic information.

System action

The protected conversation allocate request was unsuccessful. Until the mismatch is resolved, no protected conversations between the local and partner LU can be allocated.

Additional diagnostic information is written to the logrec data set to assist in diagnosing the problem.

System programmer response

To resolve the warm/cold mismatch, see [z/OS MVS Planning: APPC/MVS Management](#).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB80130I **APPC/MVS detected a log name mismatch during an Exchange Log Name processing.**

Explanation

While APPC/MVS was processing an Allocate call, exchange-log-name processing was required. During the exchange processing, an LU detected a log-name mismatch. Message ATB80125I accompanies this message, and ATB211E is also issued to provide additional diagnostic information.

System action

The protected conversation allocate request was unsuccessful. Until the mismatch is resolved, no protected conversations between the local and partner LU can be allocated.

Additional diagnostic information is written to the logrec data set to assist in diagnosing the problem.

System programmer response

To resolve the log name mismatch, see [z/OS MVS Planning: APPC/MVS Management](#).

Programmer response

Contact the system programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRS

ATB80131I Conversation was terminated due to a break-tree situation.

Explanation

A TP called a conversation service that cannot be processed because a conversation failure or conversation deallocation occurred in the allocation tree during a previous syncpt or backout verb. The break in the allocation tree resulted in the local conversation being deallocated as part of logical unit of work identifier (LUWID) management.

System action

The called service returns a return code of deallocated_abend_svc (decimal 30). The conversation has been deallocated and all resources associated with the conversation have been cleaned up.

Source

APPC/MVS

Detecting Module:

ATBPCCE

ATB80133I APPC data structures for the TP are in use because a Syncpoint or Backout verb is still running.

Explanation

A TP called an APPC/MVS TP conversation service. The system could not process the request because APPC/MVS data structures for the conversation are in use; a Commit (that is, Syncpoint) or Backout request was issued for a unit of recovery that includes the conversation for which the conversation service was issued.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service.

System programmer response

Using the DISPLAY APPC,TP and DISPLAY APPC,UR commands, determine whether the syncpoint operation is still in progress, and whether resynchronization is required.

Programmer response

Your TP can periodically retry the call to the conversation service; the call should be successful once the syncpoint operation completes. If this in-use condition persists, notify the system programmer.

Source

APPC/MVS

Detecting Module:

ATBVSRB

ATB80134I APPC/MVS detected a protocol violation during a syncpoint processing.

Explanation

A TP called a conversation service that cannot be processed because the conversation was deallocated during a previous Commit or Backout call. During the previous syncpoint operation, the partner LU made a protocol violation, which was reported through message ATB220I.

Detecting Module:

ATBPCPR, ATBPCBO, ATBPCDS, ATBPCCM, ATBPCEU, ATBPCCE, ATBPCEF

System action

The called service returns a return code of deallocated_abend_svc (decimal 30). The conversation has been deallocated and all resources associated with the conversation have been cleaned up.

ATB80135I APPC/MVS detected a protocol violation during Resynchronization Processing.

Explanation

During resynchronization processing, an LU detected an error in the data sent by its partner LU. Message ATB206E or ATB218E is also issued on the detecting system when this error occurs, and identifies the local and partner LUs.

System action

APPC/MVS resynchronization processing for the logical unit of work identified in message ATB214I is suspended.

Additional diagnostic information is written to the logrec data set to identify the violation.

System programmer response

Contact the designated support group for your installation.

Source

APPC/MVS

Detecting Module: ATBPCRS	
--	--

ATB80136I	Conversation was terminated because the unit of recovery was resolved by the installation before a syncpt decision was received from the syncpt initiator.
------------------	---

System action

The called service returns a return code of resource_failure_no_retry (decimal 26). The conversation has been deallocated and all resources associated with the conversation have been cleaned up.

Detecting Module: ATBPCEU	
--	--

ATB80138I	Sync_level of Syncpt is not valid when the partner LU only supports single sessions.
------------------	---

Explanation

A TP called the LU 6.2 Allocate service or CPI-C Allocate Service to allocate a conversation with a synchronization level of sync_level_syncpt. The request failed because the partner lu session capability is not parallel. APPC/MVS does not support

allocating syncpt conversations with a partner lu that does not have parallel session capability.

System action

The called service returns a return code of product_specific_error (decimal 20). The conversation was not allocated.

Detecting Module: ATBAMAL	
--	--

ATB80139I	APPC/MVS detected an error during Purge Log Name processing.
------------------	---

Explanation

A purge log name affinity (PLNA) request either initiated or received by an APPC/MVS logical unit failed.

System action

A purge log name affinity (PLNA) request failed. The system continues processing, but log name affinities between an APPC/MVS logical unit and a partner LU persist. The system writes this message to the logrec data set.

Detecting Module: ATBPCPL	
--	--

System programmer response

Use the diagnostic records written to the logrec data set to identify the reason for the failure.

ATB80140I	From Resource Recovery Services (RRS/MVS) callable service: service_name. Error Return Code: rrs_error_rc
------------------	--

Explanation

A TP called an APPC/MVS TP conversation service to process a request for a conversation with a synchronization level of syncpt. The system could not process the request because a system syncpoint manager (RRS) service failed during the processing of the conversation service request.

In the message text:

service_name

The name of the RRS service that failed

rrs_error_rc

The return code from the failing service.

System action

The system returns a product_specific_error (decimal 20) return code to the caller of the conversation service. The service completes unsuccessfully.

System programmer response

Determine the reason for the RRS service failure, using the name of the RRS service and the return code. See [z/OS MVS Programming: Resource Recovery](#) for the service and return code description.

Programmer response

Contact the System Programmer.

Source

APPC/MVS

Detecting Module:

**ATBAMRC, ATBVSAL, ATBVSCA, ATBVSGC,
ATBVSRR, ATBVSSR**

**ATB80141I Retrieved incomplete UR that is in
in-doubt state was not found in
the APPC log.**

Explanation

The contents of the APPC/MVS logstream cannot be used to resolve incomplete units of recovery in in-doubt state. The logstream may have been deleted and redefined or an internal APPC/MVS error has occurred. As a result, APPC/MVS is unable to automatically resynchronize these URs when the LU is reinitialized.

System action

The unit of recovery remains in in-doubt state until manual intervention resolves it. APPC/MVS will not perform resynchronization for this UR.

System programmer response

Go to the RRS administration panels and resolve the in-doubt UR identified by urid. For more information on how to use these panels, see [z/OS MVS Programming: Resource Recovery](#).

Programmer response

Contact the System Programmer.

Source

APPC/MVS

Detecting Module:

ATBPCRR

**ATB80142I APPC/MVS terminated the
conversation because APPCCMD
did not complete in the time-out
limit specified by the caller.**

Explanation

A TP called an APPC/MVS conversation callable service that resulted in an invocation of the VTAM APPCCMD macro. A timeout_value was specified for the conversation using the Allocate or Set_Timeout_Value service. The APPCCMD request did not complete in the time limit specified. APPC/MVS terminated the session to resolve a possible hang in the VTAM APPCCMD processing. Termination of the session also resulted in termination of the conversation.

System action

The system returns a Resource_Failure_Retry (decimal 26) or Resource_Failure_Retry_BO (decimal 133, for SyncLevel=Syncpt conversation) return code to the caller of the specified conversation service. If a Resource_Failure_Retry or Resource_Failure_Retry_BO return code cannot be returned for the specified conversation service, then the system returns a Product_Specific_Error (decimal 20) return code.

Programmer response

In some cases the delay in APPCCMD processing is expected. For example, a TP expects the Receive_And_Wait service to hang because the partner TP sends data intermittently. In this situation abnormal termination of the session may not be desirable. Change the application either to remove a call to the Set_Timeout_Value service or to specify a higher Timeout_Value to match the expected delay. See [“Allocate” on page 125](#) or [“Set_TimeOut_Value” on page 254](#) for an explanation of the Timeout_Value parameter.

Source

APPC/MVS

Detecting Module:

**ATBAMAL, ATBAMCD, ATBAMCF, ATBAMDE,
ATBAMEL, ATBAMFL, ATBAMPT, ATBAMRE,
ATBAMSD, ATBAMSR**

ATB80143I **Value specified
on Timeout_Value_Seconds
parameter is not valid.**

Explanation

A TP called the LU 6.2 Allocate or Set_TimeOut_Value service. The Timeout_Value_Seconds parameter specified a value that is not valid.

System action

The system returns a Program_Parameter_Check (decimal 24) return code to the caller of the specified conversation service.

Programmer response

See “Allocate” on page 125 or “Set_TimeOut_Value” on page 254 for an explanation of the Timeout_Value_Seconds parameter. Specify a valid value on the Timeout_Value_Seconds parameter.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSST

ATB80144I **The APPC Conversational Timer
monitor is not available.**

Explanation

A TP called the LU 6.2 Allocate or Set_TimeOut_Value service. The APPC Conversational Timer Monitor is not available, due to prior internal errors in the system.

System action

The system returns a Product_Specific_Error (decimal 20) return code to the caller of the specified conversation service.

System programmer response

Go to the RRS administration panels and resolve the in-doubt UR identified by urid. For more information on how to use these panels, see [z/OS MVS Programming: Resource Recovery](#).

Programmer response

Contact the System Programmer.

Source

APPC/MVS

Detecting Module:
ATBVSAL, ATBVSST

ATB80145I **APPC/MVS did not permit this
conversation to start because
the maximum number of
conversations has already been
reached by this address space.**

Explanation

The APPC active conversations threshold specified by the CONVMAX parameter has been reached. No new conversations will be allowed to start in this address space until conversations have been deallocated or cleaned up.

For each APPC active conversation on the system, APPC reserves a certain amount of system storage. A runaway transaction program, which creates many conversations but never deallocates them, could potentially exhaust the fixed amount of this system storage that APPC has obtained. To inform the installation of such a problem and to optionally prevent new conversations from being started up until the problem is rectified, APPC allows the installation to specify a threshold which will cause APPC to take action when the system encounters this problem.

System action

The system prohibits new conversations from starting in the address space.

System programmer response

You can define the APPC active conversation threshold limit on the CONVMAX subparameter of the PARM parameter of the EXEC statement in the APPC member of SYS1.PROCLIB. The value of CONVMAX must be a 1-digit to 5-digit number indicating the maximum number of APPC active conversations allowed in a single address space before APPC prevents new conversations from starting in the address space, if CMACTION has been set to HALTNEW.

Programmer response

The system did not permit this conversation to start. Check to see if you have other conversations in this

address space that were never deallocated properly when the program was finished with them. If there are conversations that should have been deallocated, modify the design of your program to deallocate the conversations when the exchange of APPC data has completed.

Source

APPC/MVS

Detecting Module:

ATBVSAL, ATBVPSA

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 40. 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 40. 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 40. 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. Explanations of Return Codes for CPI Communications Services

Table 41 on page 391 describes return codes that can be returned to a caller of a CPI Communications service.

Table 41. Return Codes for CPI Communications Services	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
1	<p>Value: Allocate_failure_no_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> Virtual telecommunications access method (VTAM) could not establish a session with the partner LU. APPC/MVS could not establish a conversation. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p> <p>See <i>z/OS Communications Server: SNA Programmer's LU 6.2 Guide</i> for a description of the sense codes included in the message from Error_Extract. If the error persists, verify that the name specified on the Local_LU_name parameter is correct. If the name is correct, contact the system programmer.</p> <p>System Programmer Response: At the request of the application programmer, ensure that the local LU is defined correctly in the VTAM application (APPL) statement in SYS1.VTAMLST.</p>
2	<p>Value: Allocate_failure_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. The system cannot allocate the conversation because of a condition that might be temporary.</p> <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: Retry the allocate request.</p>

Table 41. Return Codes for CPI Communications Services (continued)	
Return Code	Value, Meaning, and Action
3	<p>Value: Conversation_type_mismatch</p> <p>Meaning: The partner LU rejected an allocate request. The local TP called the Allocate service and specified a value of Basic_conversation or Mapped_conversation on the Conversation_type parameter. The partner TP does not support the respective basic or mapped conversation protocol boundary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: When requesting the allocate, change the Conversation_type parameter to specify a conversation type that the partner TP supports.</p>
6	<p>Value: Security_not_valid</p> <p>Meaning: The partner LU rejected an allocate request. The specified security information is not valid.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
8	<p>Value: Sync_lvl_not_supported_pgm</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a synchronization level (on the Sync_level parameter) that the partner TP does not support.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See “Allocate” on page 125 for an explanation of the Sync_level parameter. When requesting the allocate, ensure that the Sync_level parameter specifies a correct value.</p>
9	<p>Value: TPN_not_recognized</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU does not recognize.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
10	<p>Value: TP_not_available_no_retry</p> <p>Meaning: The partner LU rejected an Allocate request. The local TP specified a partner TP that is known to the partner LU, but the partner LU cannot start the TP. The condition is not temporary. The TP should not retry the Allocate request.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 41. Return Codes for CPI Communications Services (continued)	
Return Code	Value, Meaning, and Action
11	<p>Value: TP_not_available_retry</p> <p>Meaning: The partner LU rejected an allocate request. The local TP specified a partner TP that the partner LU recognizes but cannot start. The condition might be temporary.</p> <p>System Action: The system returns this return code on a call that occurs after the call to Allocate.</p> <p>Application Programmer Response: Retry the allocate request. If the error persists, see Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
17	<p>Value: Deallocated_abend</p> <p>Meaning: A partner TP called the Deallocate service. The request specified a Deallocate_type of Deallocated_abend.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges information sent by the local TP that was not received by the partner TP. The system returns this return code to the local TP when it calls an APPC service in Send or Receive state.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
18	<p>Value: Deallocated_normal</p> <p>Meaning: A partner TP called the Deallocate service for a basic or mapped conversation. The request specified a Deallocate_type of Deallocate_sync_level or Deallocate_flush.</p> <p>System Action: The system returns this return code to the local TP when it calls a service while the conversation is in Receive state.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
19	<p>Value: Parameter_error</p> <p>Meaning: A local TP called an APPC service. A parameter specified on the call is not valid. One of the following occurred:</p> <ul style="list-style-type: none"> • SNASVCMG was specified as the mode name, or the mode name was not valid. • X'0E' or X'0F' was used as the first character of a TP name. • X'06' was used as the first character of a TP name by a caller that was not running either in supervisor state or with PSW key 0-7. • An SNA service TP name was used with a mapped conversation service. <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 41. Return Codes for CPI Communications Services (continued)

Return Code	Value, Meaning, and Action
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
21	<p>Value: Program_error_no_trunc</p> <p>Meaning: Indicates one of the following:</p> <ul style="list-style-type: none"> • A partner TP called the Send_Error service for a mapped conversation. The conversation for the local TP was in Send state. No truncation occurs at the mapped conversation protocol boundary. • A partner TP called Send_Error for a basic conversation. The conversation was in Send state. The call did not truncate a logical record. No truncation occurs at the basic conversation protocol boundary when a TP calls Send_Error either before sending any logical records or after sending a complete logical record. <p>System Action: The system returns this return code to the local TP when it calls the Receive service, before the TP receives any data records or after it receives one or more data records.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
22	<p>Value: Program_error_purging</p> <p>Meaning: A partner TP called the Send_Error service for a basic or mapped conversation. The conversation for the partner TP was in Receive or Confirm state.</p> <p>System Action: The system returns this return code to the local TP when it calls an APPC service before sending any information. If the TP called Send_Error while in Receive state and before it received all the information that the partner TP sent, the system might purge the data. If the TP called Send_Error while in Receive or Confirm state but after it received all the information that the partner TP sent, the system does not purge the data.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
23	<p>Value: Program_error_trunc</p> <p>Meaning: The partner TP called the Send_Error service for a basic conversation. The conversation for the partner TP was in Send state, and the call truncated a logical record. Truncation occurs at the basic conversation protocol boundary when a TP begins sending a logical record and then makes a Send_error call before sending the complete logical record.</p> <p>System Action: The system returns this return code to the local TP on a Receive call that occurs after the TP receives the truncated logical record.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 41. Return Codes for CPI Communications Services (continued)	
Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: For the CPI Communications Initialize_Conversation service, the SYMDEST name was not found in the side information.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: The local TP called a service while running in a state in which the call is not valid. The TP should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged. For a list of states that are valid for each call, see Appendix C, “APPC/MVS Conversation State Table,” on page 399.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
26	<p>Value: Resource_failure_no_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
27	<p>Value: Resource_failure_retry</p> <p>Meaning: An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Retry the transaction.</p>
28	<p>Value: Unsuccessful</p> <p>Meaning: A call to a TP conversation service was not successful.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 41. Return Codes for CPI Communications Services (continued)

Return Code	Value, Meaning, and Action
30	<p>Value: Deallocated_abend_SVC</p> <p>Meaning: The partner TP called Deallocate with a Deallocate_type of Deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
31	<p>Value: Deallocated_abend_timer</p> <p>Meaning: A partner TP called the Deallocate service with a Deallocate_type of Deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
130	<p>Value: Deallocated_abend_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. The partner program issued a Deallocate call with Deallocate_type set to deallocate_abend, or the partner LU has done so because of a partner program abnormal-end condition.</p> <p>System Action: If the conversation for the partner program was in Receive state when the call was issued, information sent by the local program and not yet received by the partner program is purged. The conversation is now in Reset state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 41. Return Codes for CPI Communications Services (continued)	
Return Code	Value, Meaning, and Action
131	<p>Value: Deallocated_abend_SVC_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. The partner TP called Deallocate with a Deallocate_type of deallocate_abend_SVC.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
132	<p>Value: Deallocated_abend_timer_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. A partner TP called the Deallocate service with a Deallocate_type of deallocate_abend_timer.</p> <p>System Action: If the partner TP was in Receive state when it called Deallocate, the system purges all information that was sent by the local TP but was not yet received by the partner TP. The system returns this return code to the local TP when it calls a service while in Send or Receive state.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
133	<p>Value: Resource_failure_no_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. An error caused the conversation to terminate. The condition is not temporary. The application should not try to run the transaction until the condition is corrected.</p> <p>The system terminates the conversation. Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>
134	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. An error caused the conversation to terminate. The condition might be temporary.</p> <p>System Action: The system terminates the conversation.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Table 41. Return Codes for CPI Communications Services (continued)

Return Code	Value, Meaning, and Action
135	<p>Value: Resource_failure_retry_bo</p> <p>Meaning: This return code is returned only for conversations with Sync_level set to syncpt. When the Send_Error call is issued in Receive state, incoming information is purged by the system. This purged information might include an abend deallocation notification from the partner program or system. The conversation is now in Reset state.</p> <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Appendix C. APPC/MVS Conversation State Table

The APPC/MVS conversation state table shows when and where different APPC/MVS TP conversation calls can be issued. For example, a program must issue an ATBALLC call to allocate a conversation before issuing an ATBSEND call to send data.

Any APPC/MVS callable services (ATBxxxxx) that are not listed here are unrestricted by conversation state.

To see the conversation state table for the CPI Communications calls (CMxxxx) that you can issue under APPC/MVS, refer to the *CPI-C Reference*. APPC/MVS supports the states and transitions for the CPI calls as documented in that document.

When a program is in the Backout-required state, which is not a conversation state because it applies to all of the program's protected resources, the program should issue a Backout call. Until it issues a Backout call, the program will be unable to issue any of the following calls for any of its conversations with sync_level set to Syncpt that are associated with the current context. If the program issues any of these calls, the Program_state_check return code will be returned:

- Allocate
- Confirm
- Confirmed
- Flush
- Prepare_To_Receive
- Receive
- Request_To_Send
- Send_Data
- Send_Error

As described in Chapter 2, “Designing and Writing an APPC/MVS Transaction Program,” on page 21, APPC/MVS uses the concepts of states and state transitions to simplify explanations of the restrictions that are placed on the calls. A number of states are defined for APPC/MVS and, for any given call, a number of transitions are allowed. Table 42 on page 406 shows the state table, which describes the state transitions that are allowed for the APPC/MVS conversation calls.

Explanation of State-Table Abbreviations

Abbreviations are used in the state table to indicate the different permutations of calls and characteristics. There are three categories of abbreviations:

- **Conversation characteristic** abbreviations are enclosed by parenthesis — “(…)”
- **return_code** abbreviations are enclosed by brackets — “[…]”
- **data_received and status_received** abbreviations are enclosed by braces and separated by a comma — “{...,...}” — where the abbreviation before the comma represents the *data_received* value and the abbreviation after the comma represents the value of *status_received*.

The next sections show the abbreviations used in each category.

Conversation Characteristics ()

The following abbreviations are used for conversation characteristics:

Abbreviation	Meaning
A	<i>deallocate_type</i> is set to DEALLOCATE_ABEND

Abbreviation	Meaning
B	<i>send_type</i> is set to BUFFER_DATA
C	<p>For a Deallocate call, C means one of the following:</p> <ul style="list-style-type: none"> • <i>deallocate_type</i> is set to DEALLOCATE_CONFIRM • <i>deallocate_type</i> is set to DEALLOCATE_SYNC_LVL and <i>sync_level</i> is set to CONFIRM. <p>For a Prepare_To_Receive, C means one of the following:</p> <ul style="list-style-type: none"> • <i>prepare_to_receive_type</i> is set to PREP_TO_RECEIVE_CONFIRM • <i>prepare_to_receive_type</i> is set to PREP_TO_RECEIVE_SYNC_LEVEL and <i>sync_level</i> is set to CONFIRM <p>For a Send_Data call, C means the following:</p> <ul style="list-style-type: none"> • <i>send_type</i> is set to SEND_AND_CONFIRM
D	<i>send_type</i> is set to SEND_AND_DEALLOCATE.
F	<p>For a Deallocate call, F means one of the following:</p> <ul style="list-style-type: none"> • <i>deallocate_type</i> is set to DEALLOCATE_FLUSH • <i>deallocate_type</i> is set to DEALLOCATE_SYNC_LEVEL and <i>sync_level</i> is set to NONE. <p>For a Prepare_To_Receive call, F means one of the following:</p> <ul style="list-style-type: none"> • <i>prepare_to_receive_type</i> is set to PREP_TO_RECEIVE_FLUSH • <i>prepare_to_receive_type</i> is set to PREP_TO_RECEIVE_SYNC_LEVEL and <i>sync_level</i> is set to NONE. <p>For a Send_Data call, F means the following:</p> <ul style="list-style-type: none"> • <i>send_type</i> is set to SEND_AND_FLUSH
P	<i>send_type</i> is set to SEND_AND_PREP_TO_RECEIVE
S	<p>For a Deallocate call, S means:</p> <ul style="list-style-type: none"> • <i>deallocate_type</i> is set to DEALLOCATE_SYNC_LVL and <i>sync_level</i> is set to Syncpt. <p>For a Prepare_To_Receive call, S means:</p> <ul style="list-style-type: none"> • <i>prepare_to_receive_type</i> is set to PREP_TO_RECEIVE_SYNC_LEVEL and <i>sync_level</i> is set to Syncpt.
*	For a Send_Data call, * means the characteristics can be B, C, D, F, or P.

Return Code Values []

The following abbreviations are used for return codes:

Abbreviation	Meaning
ae	<p>For an Allocate call, ae means one of the following:</p> <ul style="list-style-type: none"> • ALLOCATE_FAILURE_NO_RETRY • ALLOCATE_FAILURE_RETRY <p>For any other call, ae means one of the following:</p> <ul style="list-style-type: none"> • CONVERSATION_TYPE_MISMATCH • PIP_NOT_SPECIFIED_CORRECTLY • SECURITY_NOT_VALID • SYNC_LEVEL_NOT_SUPPORTED_PGM • TPN_NOT_RECOGNIZED • TP_NOT_AVAILABLE_NO_RETRY • TP_NOT_AVAILABLE_RETRY
bo	<ul style="list-style-type: none"> • When returned on conversation calls, bo means (CM_)TAKE_BACKOUT. This return code is returned only for conversations with sync_level set to Syncpt. • When returned on the Commit call, bo means RR_BACKED_OUT.
bom	bom is a return code for Commit and Backout. It means RR_BACKED_OUT_OUTCOME_MIXED.
bop	bop is a return code for Commit and Backout. It means RR_BACKED_OUT_OUTCOME_PENDING.
com	com is a return code for Commit. It means RR_COMMITTED_OUTCOME_MIXED.
cop	cop is a return code for Commit. It means RR_COMMITTED_OUTCOME_PENDING.
sc	sc is a return code for Commit. It means PROGRAM_STATE_CHECK.
da	<p>da means one of the following:</p> <ul style="list-style-type: none"> • DEALLOCATED_ABEND • DEALLOCATED_ABEND_SVC • DEALLOCATED_ABEND_TIMER
db	<p>db is returned only for conversations with sync_level set to Syncpt and means one of the following:</p> <ul style="list-style-type: none"> • DEALLOCATED_ABEND_BO • DEALLOCATED_ABEND_SVC_BO • DEALLOCATED_ABEND_TIMER_BO
dn	DEALLOCATED_NORMAL
dnb	DEALLOCATED_NORMAL_BO. This return code is returned only for conversations with sync_level set to Syncpt.
en	<p>en means one of the following:</p> <ul style="list-style-type: none"> • PROGRAM_ERROR_NO_TRUNC • SVC_ERROR_NO_TRUNC

Abbreviation	Meaning
ep	ep means one of the following: <ul style="list-style-type: none"> • PROGRAM_ERROR_PURGING • SVC_ERROR_PURGING
et	et means one of the following: <ul style="list-style-type: none"> • PROGRAM_ERROR_TRUNC • SVC_ERROR_TRUNC
ok	OK
pe	PARAMETER_ERROR
pc	PROGRAM_PARAMETER_CHECK
rb	rb is returned only for conversations with sync_level set to Syncpt and means one of the following: <ul style="list-style-type: none"> • RESOURCE_FAILURE_NO_RETRY_BO • RESOURCE_FAILURE_RETRY_BO
rf	RESOURCE_FAILURE_NO_RETRY or RESOURCE_FAILURE_RETRY
un	UNSUCCESSFUL

Note: The return code PRODUCT_SPECIFIC_ERROR is not included in the state table because the state transitions caused by this return code are based on the environment in which the specific error is encountered. The TP may issue the APPC/MVS Error_Extract service to help diagnose the problem that was encountered by APPC/MVS.

Data_received and Status_received {, }

The following abbreviations are used for the *data_received* values:

Abbreviation	Meaning
dr	Means one of the following: <ul style="list-style-type: none"> • DATA_RECEIVED • COMPLETE_DATA_RECEIVED • INCOMPLETE_DATA_RECEIVED
nd	NO_DATA_RECEIVED
*	Means one of the following: <ul style="list-style-type: none"> • DATA_RECEIVED • COMPLETE_DATA_RECEIVED • NO_DATA_RECEIVED

The following abbreviations are used for the *status_received* values:

Abbreviation	Meaning
cd	CONFIRM_DEALLOC_RECEIVED
cs	CONFIRM_SEND_RECEIVED
co	CONFIRM_RECEIVED

Abbreviation	Meaning
no	NO_STATUS_RECEIVED
se	SEND_RECEIVED
tc	TAKE_COMMIT. This value is returned only for conversations with sync_level set to Syncpt.
td	TAKE_COMMIT_DEALLOCATE. This value is returned only for conversations with sync_level set to Syncpt.
ts	TAKE_COMMIT_SEND. This value is returned only for conversations with sync_level set to Syncpt.

Table Symbols

The following symbols are used in the state table to indicate the condition that results when a call is issued from a certain state:

Symbol	Meaning
–	Remain in current state
1-8	Number of next state
>	State error. A <i>return_code</i> of PROGRAM_STATE_CHECK is returned. For calls illegally issued in Reset state, this condition is indicated to the program with a return code of PROGRAM_PARAMETER_CHECK. This is because the program is in Reset state and the <i>conversation_ID</i> for the conversation is undefined.
^	For programs not using sync_level set to Syncpt, this symbol should be ignored. For programs using sync_level set to Syncpt, when this symbol follows a state number or a – (for example, 1^ or –^), it means the program may be in the Backout-Required condition following the call.
↑	For programs not using sync_level set to Syncpt, this symbol should be ignored. For programs using sync_level set to Syncpt, when this symbol follows the name of a call (for example, ATBDEAL(F)[ok]↑), it means that it is valid to make this call (for example, ATBDEAL(F)) as indicated in the state transition table unless the program is in the Backout-required condition. In that case, the call is invalid and PROGRAM_STATE_CHECK is returned.
#	Conversations with sync_level set to Syncpt go to the state they were in at the completion of the most recent synchronization point. If there was no prior synchronization event, the side of allocator goes to Send state, and the allocatee goes to the Receive state.

How to Use the State Table

The various calls and combinations of parameters, also referred to as *inputs*, are shown along the left side of the table. These inputs correspond to the rows of the table. The possible states are shown across the top of the table. The states correspond to the columns of the matrix. The intersection of input (row) and state (column) represents what state transition, if any, will occur for the APPC/MVS call that is issued in that particular state.

For example, look at ATBALLC[ok]. The [ok] indicates that a return code of OK was received on the call. By examining the row, notice that the call can be issued only in **Reset** state (state 1).

When issued in state 1, the 3 in the column for **Reset** indicates that the program progresses to state 3, **Send**. A scan down this column shows that the ATBSEND call can be made from here. Some variations in the ATBSEND row that entail a change of state are for return codes of:

State Table

- “ok” when the conversation characteristic (P) indicates that the *send_type* is set to SEND_AND_PREP_TO_RECEIVE, which allows the program to progress to state 4.
- “ok” when the conversation characteristic (D) indicates that the *send_type* is set to SEND_AND_DEALLOCATE, which puts the program back into Reset state (state 1).

Table 42. States and Transitions for APPC/MVS Conversation Calls

Inputs	Used by all conversations									Used only by conversations with sync_level set to Syncpt			
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBALC2[ok] ↑	3	>	>	>	>	>	>	>	>	>	>	>	>
ATBALC2[ae]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALC2[pe]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALC2[pc]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALC2[un]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALLC[ok] ↑	3	>	>	>	>	>	>	>	>	>	>	>	>
ATBALLC[ae]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALLC[pe]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALLC[pc]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBALLC[un]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBCFM[ok] ↑	>	>	—	>	3	>	>	>	4	>	>	>	>
ATBCFM[ae]	>	>	1	>	1	>	>	>	1	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial- ize 2	Send 3	Re- ceive 4	Send Pend- ing 5	Con- firm 6	Con- firm Send 7	Con- firm Deal - locate 8	Defer- Re- ceive 9	Defer- Deal- locate 10	Sync- Point 11	Sync- Point Send 12	Sync- Point Deal- locate 13
ATBCFM[da]	>	>	1	>	1	>	>	>	1	>	>	>	>
ATBCFM[ep]	>	>	4	>	4	>	>	>	4	>	>	>	>
ATBCFM[rf]	>	>	1	>	1	>	>	>	1	>	>	>	>
ATBCFM[pc]	>	>	-	>	-	>	>	>	-	>	>	>	>
ATBCFM[bo]	>	>	-^	>	3^	>	>	>	4^	>	>	>	>
ATBCFM[rb]	>	>	1^	>	1^	>	>	>	1^	>	>	>	>
ATBCFM[db]	>	>	1^	>	1^	>	>	>	1^	>	>	>	>
ATBCFMD[ok] ↑	>	>	>	>	>	4	3	1	>	>	>	>	>
ATBCFMD[pc]	>	>	>	>	>	-	-	-	>	>	>	>	>
ATBDEAL(F)[ok] ↑	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBDEAL(F)[pc]	>	>	-	>	-	>	>	>	>	>	>	>	>
ATBDEAL(C)[ok]	>	>	1	>	1	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send- Pend- ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal- locate 8	Defer- Re- ceive 9	Defer- Deal- locate 10	Sync- Point 11	Sync- Point Send 12	Sync- Point Deal- locate 13
ATBDEAL(C)[ae]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBDEAL(C)[da]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBDEAL(C)[ep]	>	>	4	>	4	>	>	>	>	>	>	>	>
ATBDEAL(C)[rf]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBDEAL(C)[pc]	>	>	—	>	—	>	>	>	>	>	>	>	>
ATBDEAL(A)[ok]	>	>	1^	1^	1^	1^	1^	1^	1^	1^	1^	1^	1^
ATBDEAL(A)[pc]	>	>	—	—	—	—	—	—	—	—	—	—	—
ATBDEAL(S)[ok] ↑	>	>	10	>	10	>	>	>	>	>	>	>	>
ATBDEAL(S)[pc]	>	>	—	>	—	>	>	>	>	>	>	>	>
ATBFLUS[ok] ↑	>	>	—	>	3	>	>	>	4	>	>	>	>
ATBFLUS[pc]	>	>	—	>	—	>	>	>	—	>	>	>	>
ATBGETC[ok]	4	>	>	>	>	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)													
Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBGETC[pc]	—	>	>	>	>	>	>	>	>	>	>	>	>
ATBGTA2	>	>	—	—	—	—	—	—	—	—	—	—	—
ATBGETA	>	>	—	—	—	—	—	—	—	—	—	—	—
ATBPOR2	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBPTR(F)[ok] ↑	>	>	4	>	4	>	>	>	>	>	>	>	>
ATBPTR(F)[pc]	>	>	—	>	—	>	>	>	>	>	>	>	>
ATBPTR(S)[ok] ↑	>	>	9	>	9	>	>	>	>	>	>	>	>
ATBPTR(S)[pc]	>	>	—	>	—	>	>	>	>	>	>	>	>
ATBPTR(C)[ok] ↑	>	>	4	>	4	>	>	>	>	>	>	>	>
ATBPTR(C)[ae]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBPTR(C)[da]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBPTR(C)[ep]	>	>	4	>	4	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBPTR(C)[rf]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBPTR(C)[pc]	>	>	-	>	-	>	>	>	>	>	>	>	>
ATBPTR(C)[bo]	>	>	4^	>	4^	>	>	>	>	>	>	>	>
ATBPTR(C)[db]	>	>	1^	>	1^	>	>	>	>	>	>	>	>
ATBPTR(C)[rb]	>	>	1^	>	1^	>	>	>	>	>	>	>	>
ATBRCVI[ok]{dr,no}↑	>	>	>	-	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{nd,se}	>	>	>	3	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{dr,se}	>	>	>	5	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{*,co}	>	>	>	6	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{*,cs}	>	>	>	7	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{*,cd}	>	>	>	8	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{*,tc}	>	>	>	11	>	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)													
Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial- ize 2	Send 3	Re- ceive 4	Send Pend- ing 5	Con- firm 6	Con- firm Send 7	Con- firm Deal - locate 8	Defer- Re- ceive 9	Defer- Deal- locate 10	Sync- Point 11	Sync- Point Send 12	Sync- Point Deal- locate 13
ATBRCVI[ok]{*,ts}	>	>	>	12	>	>	>	>	>	>	>	>	>
ATBRCVI[ok]{*,td}	>	>	>	13	>	>	>	>	>	>	>	>	>
ATBRCVI[ae] ↑	>	>	>	1	>	>	>	>	>	>	>	>	>
ATBRCVI[da]	>	>	>	1	>	>	>	>	>	>	>	>	>
ATBRCVI[dn]	>	>	>	1	>	>	>	>	>	>	>	>	>
ATBRCVI[en]	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBRCVI[ep]	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBRCVI[et]	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBRCVI[rf]	>	>	>	1	>	>	>	>	>	>	>	>	>
ATBRCVI[pc]	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBRCVI[un]	>	>	>	—	>	>	>	>	>	>	>	>	>
ATBRCVI[bo]	>	>	>	✓	>	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBRCVI[db]	>	>	>	1^	>	>	>	>	>	>	>	>	>
ATBRCVI[rb]	>	>	>	1^	>	>	>	>	>	>	>	>	>
ATBRCVW[ok]{dr,no} ↑	>	>	4	—	4	>	>	>	>	>	>	>	>
ATBRCVW[ok]{nd,se}	>	>	—	3	3	>	>	>	>	>	>	>	>
ATBRCVW[ok]{dr,se}	>	>	5	5	—	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,co}	>	>	6	6	6	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,cs}	>	>	7	7	7	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,cd}	>	>	8	8	8	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,tc}	>	>	11	11	11	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,ts}	>	>	12	12	12	>	>	>	>	>	>	>	>
ATBRCVW[ok]{*,td}	>	>	13	13	13	>	>	>	>	>	>	>	>
ATBRCVW[ae] ↑	>	>	1	1	1	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)													
Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pending 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBRCVW[da]	>	>	1	1	1	>	>	>	>	>	>	>	>
ATBRCVW[dn]	>	>	1	1	1	>	>	>	>	>	>	>	>
ATBRCVW[en]	>	>	4	—	4	>	>	>	>	>	>	>	>
ATBRCVW[ep]	>	>	4	—	4	>	>	>	>	>	>	>	>
ATBRCVW[et]	>	>	>	—	4	>	>	>	>	>	>	>	>
ATBRCVW[rf]	>	>	1	1	1	>	>	>	>	>	>	>	>
ATBRCVW[pc]	>	>	—	—	—	>	>	>	>	>	>	>	>
ATBRCVW[bo]	>	>	4^	—^	4^	>	>	>	>	>	>	>	>
ATBRCVW[db]	>	>	1^	1^	1^	>	>	>	>	>	>	>	>
ATBRCVW[rb]	>	>	1^	1^	1^	>	>	>	>	>	>	>	>
ATBRJC2	>	>	>	1	>	>	>	>	>	>	>	>	>
ATBRTS[ok] ↑	>	>	—	—	—	—	—	—	>	>	—	—	—

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)														
Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt					
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13	
ATBRTS[pc]	>	>	-	-	-	-	-	-	>	>	-	-	-	-
ATBSCA2	>	-	-	-	-	-	-	-	-	-	-	-	-	-
ATBSEND(B)[ok] ↑	>	>	-	>	3	>	>	>	>	>	>	>	>	>
ATBSEND(F)[ok]	>	>	-	>	3	>	>	>	>	>	>	>	>	>
ATBSEND(C)[ok]	>	>	-	>	3	>	>	>	>	>	>	>	>	>
ATBSEND(P(C))[ok]	>	>	4	>	4	>	>	>	>	>	>	>	>	>
ATBSEND(P(F))[ok]	>	>	4	>	4	>	>	>	>	>	>	>	>	>
ATBSEND(P(S))[ok]	>	>	9	>	9	>	>	>	>	>	>	>	>	>
ATBSEND(D(A))[ok]	>	>	1^	>	1^	>	>	>	>	>	>	>	>	>
ATBSEND(D(C))[ok]	>	>	1	>	1	>	>	>	>	>	>	>	>	>
ATBSEND(D(F))[ok]	>	>	1	>	1	>	>	>	>	>	>	>	>	>
ATBSEND(D(S))[ok]	>	>	10	>	10	>	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send- Pend- ing 5	Con- firm 6	Con- firm Send 7	Con- firm Deal- locate 8	Defer- Re- ceive 9	Defer- Deal- locate 10	Sync- Point 11	Sync- Point Send 12	Sync- Point Deal- locate 13
ATBSEND(*)[ae] ↑	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBSEND(*)[da]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBSEND(*)[ep]	>	>	4	>	4	>	>	>	>	>	>	>	>
ATBSEND(*)[rf]	>	>	1	>	1	>	>	>	>	>	>	>	>
ATBSEND(*)[pc]	>	>	—	>	—	>	>	>	>	>	>	>	>
ATBSEND(*)[bo]	>	>	— [^]	>	3 [^]	>	>	>	>	>	>	>	>
ATBSEND(*)[db]	>	>	1 [^]	>	1 [^]	>	>	>	>	>	>	>	>
ATBSEND(*)[rb]	>	>	1 [^]	>	1 [^]	>	>	>	>	>	>	>	>
ATBSERR[ok] ↑	>	>	—	3	3	3	3	3	>	>	3	3	3
ATBSERR[ae]	>	>	1	>	>	>	>	>	>	>	>	>	>
ATBSERR[da]	>	>	1	>	>	>	>	>	>	>	>	>	>
ATBSERR[dn]	>	>	>	1	>	>	>	>	>	>	>	>	>

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)

Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
ATBSERR[ep]	>	>	4	>	>	>	>	>	>	>	>	>	>
ATBSERR[rf]	>	>	1	1	1	1	1	1	>	>	1	1	1
ATBSERR[pc]	>	>	-	-	-	-	-	-	>	>	-	-	-
ATBSERR[bo]	>	>	-	>	3^	>	>	>	>	>	-	-	-
ATBSERR[rb]	>	>	1^	1^	1^	1^	1^	1^	>	>	1^	1^	1^
ATBSERR[db]	>	>	1^	>	>	>	>	>	>	>	>	>	>
ATBSERR[dnb]	>	>	>	1^	>	>	>	>	>	>	>	>	>
SRRCMIT[ok]	-	-	-	>	3	>	>	>	4	1	4	3	1
SRRCMIT[cop]	-	-	-	>	3	>	>	>	4	1	4	3	1
SRRCMIT[com]	-	-	-	>	3	>	>	>	4	1	4	3	1
SRRCMIT[bo]	-	-	#	>	#	>	>	>	#	#	#	#	#
SRRCMIT[bop]	-	-	#	>	#	>	>	>	#	#	#	#	#

Table 42. States and Transitions for APPC/MVS Conversation Calls (continued)													
Inputs	Used by all conversations								Used only by conversations with sync_level set to Syncpt				
	Re-set 1	Ini-tial-ize 2	Send 3	Re-ceive 4	Send Pend-ing 5	Con-firm 6	Con-firm Send 7	Con-firm Deal-locate 8	Defer-Re-ceive 9	Defer-Deal-locate 10	Sync-Point 11	Sync-Point Send 12	Sync-Point Deal-locate 13
SRRCMIT[bom]	—	—	#	>	#	>	>	>	#	#	#	#	#
SRRCMIT[sc]	—	—	—	>	—	>	>	>	—	—	—	—	—
SRRBACK[ok]	—	—	#	#	#	#	#	#	#	#	#	#	#
SRRBACK[bop]	—	—	#	#	#	#	#	#	#	#	#	#	#
SRRBACK[bom]	—	—	#	#	#	#	#	#	#	#	#	#	#

Note:

1. For all SRRCMIT inputs: When a program started by an incoming allocation request issues a Commit call before issuing an Accept_Conversation or Get_Conversation for the conversation that started the program, the Commit call has no effect on the conversation in **Reset** state. This behavior is different from what the *CPI-C Reference* defines concerning the affect of issuing a Commit call against a conversation in **Reset** state. *CPI-C Reference* states that when a program started by an incoming allocation request issues a Commit call before issuing an Accept_Conversation call (the CPI equivalent to Get_Conversation), a state check results. The Commit call has no effect on other conversations in **Reset** state.
2. For all SRRBACK inputs: When a program started by an incoming allocation request issues a Backout call before issuing an Accept_Conversation or Get_Conversation for the conversation that started the program, the Backout call has no effect on the conversation in **Reset** state. This behavior is different from what the *CPI-C Reference* defines concerning the affect of issuing an SRRBACK call against a conversation in **Reset** state. *CPI-C Reference* states that when a program started by an incoming allocation request issues a Backout call before issuing an Accept_Conversation (the CPI equivalent to Get_Conversation), the underlying LU 6.2 conversation is actually backed out, though the conversation remains in **Reset** state. What this actually means is that a Backout is flowed on the unaccepted LU 6.2 conversation on the behalf of the application.
3. For all SRRCMIT and SRRBACK inputs: Conversations in **Initialize** state are not affected by Commit and Backout calls.

Appendix D. Support for SNA LU 6.2 Verbs and Option Sets

This appendix lists the APPC/MVS support for the verbs and option sets that are defined by the SNA LU 6.2 architecture.

Mapping APPC/MVS TP Services to LU 6.2 Verbs and CPI Communications

The following table lists the APPC/MVS services for transaction programs, showing the corresponding verbs from the SNA LU 6.2 architecture and the equivalent CPI Communications calls, if any.

<i>Table 43. List of APPC/MVS TP Callable Services with SNA and CPI-C Equivalents</i>		
APPC/MVS Service	SNA LU 6.2 Verb	CPI Communications Call
APPC/MVS TP Conversation Services		
ATBALC6, ATBALC5, ATBALC2 or ATBALLC	(MC_)Allocate	Initialize_Conv (CMINIT), CMALLC
ATBCFM	(MC_)Confirm	Confirm (CMCFM)
ATBCFMD	(MC_)Confirmed	Confirmed (CMCFMD)
ATBDEAL	(MC_)Deallocate	Deallocate (CMDEAL)
ATBEES3 - Error_Extract	(no SNA equivalent)	(no CPI equivalent)
ATBFLUS	(MC_)Flush	Flush (CMFLUS)
ATBGTA2 or ATBGETA	(MC_)Get_Attributes	Extr_Conv_State (CMECS), Extr_Mode_Name (CMEMN), Extr_Part_LU_Name (CMEPLN), Extr_Sync_Level (CMESL)
ATBGETC - Get_Conversation	(no SNA equivalent)	Accept_Conv (CMACCP)
ATBGETP or ATBGTP4	Get_TP_Properties	(no CPI equivalent)
ATBGETT	Get_Type	Extr_Conv_Type (CMECT)
ATBPOR2 - Post_on_Receipt	(no SNA equivalent)	(no CPI equivalent)
ATBPTR	(MC_)Prepare_to_Receive	Prepare_to_Receive (CMPTR)
ATBRCVI	(MC_)Receive_Immediate	Receive (CMRCV)
ATBRCVW	(MC_)Receive_and_Wait	Receive (CMRCV)
ATBRTS	(MC_)Request_to_Send	Req_to_Send (CMRTS)
ATBSEND	(MC_)Send_Data	Send_Data (CMSSEND)
ATBSERR	(MC_)Send_Error	Send_Error (CMSERR)
ATBSSO4	Set_Syncpt_Options	(no CPI equivalent)
ATBSTO5	(no SNA equivalent)	(no CPI equivalent)
APPC/MVS Advanced TP Services		

Table 43. List of APPC/MVS TP Callable Services with SNA and CPI-C Equivalents (continued)		
APPC/MVS Service	SNA LU 6.2 Verb	CPI Communications Call
ATBAMR1 - Asynchronous Manager service	(no SNA equivalent)	(no CPI equivalent)
ATBTEA1 - Accept_Test	(no SNA equivalent)	(no CPI equivalent)
ATBCUC1 or ATBCMCTU - Unauthorized Cleanup_TP service	(no SNA equivalent)	(no CPI equivalent)
ATBEXAI - Extract_Information	(no SNA equivalent)	(no CPI equivalent)
ATBGTRN - Obtaining the Next Transaction for Multi_Trans	(no SNA equivalent)	(no CPI equivalent)
ATBTER1 - Register_Test	(no SNA equivalent)	(no CPI equivalent)
ATBRJC2 - Reject_Conversation	(no SNA equivalent)	(no CPI equivalent)
ATBRTRN - Restoring the Multi_Trans Environment	(no SNA equivalent)	(no CPI equivalent)
ATBSCA2 - Write user data to SMF accounting record.	(no SNA equivalent)	(no CPI equivalent)
ATBTEU1 - Unregister_Test	(no SNA equivalent)	(no CPI equivalent)
ATBVERS - Version service	(no SNA equivalent)	(no CPI equivalent)

APPC/MVS Support for LU 6.2 Option Sets

This section describes the support provided by APPC/MVS for the option sets listed in the SNA LU 6.2 architecture. Option sets are LU 6.2 functions that are not required for the minimum implementation of a type 6.2 LU. The option set number is in parentheses following the option set name. For more information about the LU 6.2 option sets, see *SNA Transaction Programmer's Reference Document for LU 6.2*.

Flush the LU's Send Buffer (101)

This option set allows a program to explicitly cause the LU to transmit any data in its send buffer, regardless of the amount of data in the buffer. Support for this option set is provided in the ATBFLUS and CMFLUS communications calls.

Get Attributes (102)

This option set allows a program to obtain attributes of the conversation. Support for this option set is provided in the ATBGTA2 and ATBGETA communications calls.

Prepare to Receive (105)

This option set allows a program to change the conversation from Send state to Receive state and at the same time flush the LU's send buffer or request confirmation. Support for this option set is provided in the ATBPTR and CMPTR communications calls.

Receive Immediate (106)

This option set allows a program to receive whatever information is available on a conversation without having to request posting of the conversation. Support for this option set is provided in the ATBRCVI and CMRCV communications calls.

Sync Point Services (108)

This option set allows a program to request sync point processing for all protected resources of the transaction. Support for this option set is provided by RRS and APPC/MVS, through the SRRCMIT and SRRBACK calls, and the ATBSSO4 call, respectively.

Get Conversation Type (110)

This option set allows a program that supports both the basic conversation and mapped conversation protocol boundaries to determine which category of verbs it should use in conjunction with a resource ID. Support for this option set is provided in the ATBGETT and CMECT communications calls.

Queued Allocation of a Conwinner Session (201)

This option set allows a local program to allocate a conversation to a remote program on a session for which the local LU must be the contention winner and for which the local program will wait. Support for this option set is provided in the ATBALC2 or ATBALLC communications calls.

Immediate Allocation of a Session (203)

This option set allows a program to allocate a contention-winner session only if the session is immediately available; otherwise, the allocation is unsuccessful. Support for this option set is provided in the ATBALC2, ATBALLC, and CMALLC communications calls.

Conversations between Programs Located at the Same LU (204)

This option set allows a local program to allocate a conversation to a program at the same LU as the local program. Support for this option set is provided in the ATBALC2, ATBALLC, and CMALLC communications calls.

Session-Level LU-LU Verification (211)

Allows a program or operator to designate the LU-LU passwords, associated with remote LUs, that the local LU uses to verify the identity of a remote LU at session activation time. VTAM support for this option set is provided by the VERIFY operand of the VTAM APPL statement. RACF support for this option set is provided by the SESSKEY field of the SESSION segment of the APPCLU profile.

User ID Verification (212)

Allows a program or operator to designate the user IDs and associated passwords that the local LU uses to verify the identity of a user ID carried on allocation requests it receives, and to designate the remote LUs that are permitted to send to the local LU allocation requests carrying a user ID and either a password or an already-verified indication. Also allows the program allocating a conversation to specify that the allocation request carry the user ID received on the request that started the program, together with an already-verified indication.

VTAM support for this option set is provided by the SECACPT operand on the VTAM APPL statement. RACF support for this option set is provided by the CONVSEC field in the SESSION segment of the RACF APPCLU profile. APPC/MVS support for this option set is provided in its TP attach processing, which uses RACF services to verify the user ID and password on allocation requests it receives. Support for this option set is also provided in the ATBALC2, ATBALLC, and CMALLC communications calls.

Program Supplied User ID and Password (213)

Allows the program allocating a conversation to supply the user ID and password to be sent on the allocation request. Support for this option set is provided in the ATBALC2 and ATBALLC communications calls.

User ID Authorization (214)

Allows a program or operator to designate the user IDs that are authorized access to specific resources of the LU, such as transaction programs.

RACF support for this option set is provided by its general scheme of resource access control and the APPCTP general resource class. APPC/MVS support for this option set is provided in its TP attach processing, which creates a RACF security environment for the TP.

Profile Verification and Authorization (215)

Allows a program or operator to designate the profiles that the local LU uses to verify a profile carried on allocation requests it receives, and to designate the profiles that are authorized access to specific resources of the LU, such as transaction programs.

Support for this option set is provided in TP attach processing, which uses the profile field received on an allocation request as the RACF groupid when creating the security environment for the TP.

Origin LU Authorization (216)

APPC/MVS and RACF allow a security administrator to specify the user IDs and remote LUs that are authorized access to specific resources of a local LU. To implement origin LU authorization, you must include the APPCPORT class among the general resource classes for which conditional access lists can exist and modify RACF's PERMIT command processing. For more information about using RACF to set up origin LU authorization, see [z/OS Security Server RACF Security Administrator's Guide](#).

Profile Passthrough (217)

Allows the program allocating a conversation to specify that the allocation request carry the profile received on the request that started the program. Support for this option set is provided in the ATBALC2, ATBALLC, and CMALLC communications calls.

Program-Supplied Profile (218)

Allows the program allocating a conversation to specify the profile to be sent on the allocation request. Support for this option set is provided in the ATBALC2 and ATBALLC communications calls.

Receive Persistent Verification (220)

Allows a program or an operator to designate the remote LUs from which the local LU will accept consecutive allocation requests that, once verified, remain verified for specific user IDs.

Support for this option set is provided by APPC/MVS, ACF/VTAM (release 3.4 or later), and RACF (release 1.9.2 or later). VTAM support for this option set is provided by keyword values of the SECACPT operand on the VTAM APPL statement. RACF support for this option set is provided by keyword values of the CONVSEC field in the SESSION segment of the RACF APPCLU profile. APPC/MVS support for this option set is provided in its TP attach processing, which uses RACF services to inspect and maintain the LU's SIGNED_ON_FROM list.

Receive SIGNON/Change Password (222)

Allows a remote program or LU to sign a user on to the local LU or change the user's password, or both. If option set 220 is in use, subsequent attaches for this user can flow as "signed on." Support for this option set is provided in TP attach processing and a special TP for servicing these requests.

Accounting (243)

This option set allows an LU to generate and send both a logical-unit-of-work (LUW) identifier and a conversation correlator (CC) to the remote LU. APPC/MVS supports this option set only for protected conversations.

Long Locks (244)

This option set allows a program to perform the prepare-to-receive function and request confirmation, and resume processing when information, such as data or conversation status, is received from the remote program following an affirmative reply. Support for this option set is provided in the ATBPTR communications call.

Test for Request-to-Send Received (245)

This option set allows a program to test whether a request-to-send notification has been received on a conversation. Support for this option set is provided in the CMTRTS communications call.

Vote Read-Only Response to a Sync Point Operation (249)

This option set improves performance of sync point operations by allowing the local LU to vote read-only when none of the protected resources in its part of the distributed transaction have been changed. This option set includes the VOTE_READ_ONLY_PERMITTED parameter of the ATBSSO4 call, and affects the return code to the SRRCMIT or SRRBACK verb.

Extract Transaction and Conversation Identification Information (251)

This option set allows the TP to retrieve the information identifying the transaction and the conversations it is using. This option set relates to the Get_TP_Properties service (for the logical unit of work identifier), and the Get_TP_Attributes service (for the conversation correlator). APPC/MVS supports this option set only for protected conversations.

CHANGE_SESSION_LIMIT Verb (501)

This option set allows a program or an operator at the source LU to request a change in the (LU,mode) session limit from one nonzero value to another, or a change in the minimum number of contention-winner sessions for the source LU or target LU.

VTAM (release 3.4 or later) provides a MODIFY CNOS operator command which may be used to change the session limits for LUs operated by APPC/MVS.

Session-Level Mandatory Cryptography (611)

This option set allows a program or an operator to specify that session-level mandatory cryptography is to be used on sessions within an (LU,mode) group.

Support for this option set is provided in VTAM (release 3.4 or later). VTAM support for this option set is provided by the ENCR parameter of the MODEENT macro.

Appendix E. Previous Versions of APPC/MVS Callable Services

This section describes previous APPC/MVS TP conversation calls that have been replaced by newer versions. The newer versions are documented in [Chapter 8, “APPC/MVS TP Conversation Callable Services,”](#) on page 125. To determine the most recent version available on the system, use the Version service described in [“Version_Service”](#) on page 287. The most recent version of each service is the recommended programming interface.

ATBALLC - Allocate (For MVS/ESA 4.2 and 4.2.2)

Equivalent to:

- LU 6.2 (MC_)Allocate
- CPI Initialize_Conv (CMINIT) and Allocate (CMALLC)

This section describes the Allocate (ATBALLC) TP callable service provided with MVS/ESA Version 4 Release 2 and MVS/ESA Version 4 Release 2.2. The Allocate service was enhanced for MVS/ESA Version 4 Release 3, and renamed ATBALC2 (see [“ATBALC2 - Allocate \(For MVS/ESA 4.3 through OS/390 Release 7\)”](#) on page 433). The Allocate service was also enhanced for OS/390 Version 2 Release 8, and renamed ATBALC5 (see [“ATBALC5 - Allocate \(For OS/390 Release 8 through z/OS V1R6\)”](#) on page 448). The Allocate service was further enhanced for z/OS V1R7, and renamed ATBALC6 (see [“Allocate”](#) on page 125). The ATBALLC, ATBALC2, and ATBALC5 calls remain valid, but do not contain the enhancements included in ATBALC6.

Note: The ATBALC6 call is the recommended programming interface for this service.

Allocates a session between the local LU and a partner LU, and on that session allocates a basic or mapped conversation between the local program and a partner program. A conversation ID is assigned to the conversation. Call this service before other calls that refer to the conversation.

If the program that issues the allocate call was not started by APPC/MVS in response to an inbound allocate call, and is not associated with an alternative transaction scheduler, the outbound allocate call and ensuing conversation flow through the base LU for the APPC/MVS transaction scheduler. If, in such a case, there is no base LU defined for the APPC/MVS transaction scheduler, APPC/MVS uses the system base LU. If there is no system base LU, APPC/MVS rejects the allocate request. For more information about base LUs and their definition, see [z/OS MVS Planning: APPC/MVS Management](#).

Requirements

Authorization:	Supervisor state or problem state, any PSW key, with the following exceptions: <ul style="list-style-type: none"> • When the TP_name specified is an SNA TP name beginning with X'06', the caller must run either in supervisor state, or with PSW key 0-7. • When the TP_id specified is a value other than binary zeros, the caller must run either in supervisor state, or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB mode, with the following exception: task mode only for callers that issue Allocate for a conversation with a synchronization level of Syncpt.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit

ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBALLC(
    Conversation_type,
    Sym_dest_name,
    Partner_LU_name,
    Mode_name,
    TP_name_length,
    TP_name,
    Return_control,
    Sync_level,
    Security_type,
    User_ID,
    Password,
    Profile,
    User_Token,
    Conversation_ID,
    Notify_type,
    TP_ID,
    Return_code
);
```

Figure 65. ATBALLC - LU 6.2 Allocate

Parameters

Conversation_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the service is invoked.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

Specifies that in this conversation, the TPs will format their data into separate records, with record length and data specified, before sending it.

1

Mapped_conversation

Specifies that in this conversation, the TPs will rely on APPC to format the data that the TPs send.

Sym_dest_name

Supplied parameter

- Type: Character string
- Char Set: 01134
- Length: 8 bytes

Specifies a symbolic name representing the partner LU, the partner TP_name, and the mode name for the session on which the conversation is to be carried. The symbolic destination name must match that of an entry in the side information data set. The appropriate entry in the side information is retrieved and used to initialize the characteristics for the conversation.

If you specify a symbolic destination name, the partner LU name, mode name, and TP name are obtained from the side information. If you also specify values for the Partner_LU_name, Mode_name, or TP_name parameters on the Allocate service, these values override any obtained from the side information.

The symbolic destination name in this field can be from 1 to 8 characters long, with characters from character set 01134. If the symbolic destination name is shorter than eight characters, it must be left-justified in the variable field, and padded on the right with blanks. To not specify a symbolic destination name, set the sym_dest_name parameter value to 8 blanks and provide values for the Partner_LU_name, Mode_name, and TP_name parameters.

Partner_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes (must be padded with blanks if less than 17 bytes)

Partner_LU_name specifies the name of the LU at which the partner program is located.

The Partner_LU_name is any name by which the local LU knows the partner LU for the purposes of allocating a conversation. The local LU transforms this locally known LU name to an LU name used by the network.

The Partner_LU_name can be one of the following:

- LU name only (1-8 byte Type A character string).

This string represents the network LU name, which, if unique within the network and interconnected networks, is sufficient for most TP communications.

IBM recommends, however, that you specify either a symbolic destination name (in the Sym_dest_name parameter), or the network-qualified LU name, if known. While the network LU name might be unique currently, it might not remain so if the installation increases the number of networks in use. Specifying a symbolic destination name or network-qualified LU name can minimize the need for future network definitions and program changes.

- A VTAM generic resource name.

If the partner LU is a member of a generic resource group, you may specify the 1- to 8-byte generic resource name of the group.

- Combined network_ID and network LU name (two 1-8 byte Type A character strings, concatenated by a period: *network_ID.network_LU_name*).

This format is known as a **network-qualified LU name**; each LU in the network and all interconnected networks can be uniquely identified by its network-qualified LU name. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name. If the local LU is not enabled to support network-qualified names, APPC/MVS passes only the network-LU-name portion to VTAM, which might cause an error if the network LU name is not unique across networks.

- A value of 17 blanks:

If you specify a symbolic destination name in Sym_dest_name parameter, set Partner_LU_name to blanks to use the partner LU name from the side information.

If you do not specify a symbolic destination name, then use a blank Partner_LU_name to indicate that the partner program is located at the same LU as the local program (LU=OWN). If the local LU is defined as a member of a VTAM generic resource group, APPC/MVS uses the generic resource name for Partner_LU_name.

Mode_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes (must be padded with blanks if less than 8 bytes)

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

The mode name value of "SNASVCMG" is reserved for use by APPC/MVS. If a mode name of "SNASVCMG" is specified on the Allocate service, the request is rejected with a return code of parameter_error.

If you specify a symbolic destination name in the sym_dest_name parameter, set mode_name to blanks to obtain the mode_name from the side information.

If you do not specify a sym_dest_name and do not specify a mode name, APPC/MVS uses the default mode name "ATB#MODE".

TP_name_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

TP_name_length specifies the length of data contained in the TP_name parameter.

If you specify a symbolic destination name in the sym_dest_name parameter, set TP_name_length to 0 to use the partner TP name from the side information.

TP_name

Supplied parameter

- Type: Character string
- Char Set: 006409 (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

TP_name specifies the name of the partner program to be connected at the other end of the conversation.

If you specify a symbolic destination name in the sym_dest_name parameter and set the TP_name_length parameter to zero, the TP name is obtained from the side information file.

TP_name can specify the name of any SNA service transaction program except for one whose first character is X'06'; see the authorization requirements in ["Requirements" on page 425](#) for more information about this exception. The names of SNA service transaction programs can contain blank characters. For a list of SNA service transaction programs, see *SNA Transaction Programmer's Reference Document for LU 6.2*.

If the partner TP is to be protected by a RACF security profile in the APPCTP class, the TP_name must consist of Type A characters only.

Return_control

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_control specifies when the local LU is to return control to the local program, in relation to the allocation of a session for the conversation.

Valid values for this parameter are:

Value

Meaning

0

When_session_allocated

Specifies to allocate a session for the conversation before returning control to the program. An error in allocating a session is reported on this call.

1

Immediate

Specifies to allocate a session for the conversation if a session is immediately available, and return control to the program with a return code indicating whether a session is allocated. An error in allocating a session that is immediately available is reported on this call.

100

When_conwinner_allocated

Specifies to allocate a session in which the local LU is the contention winner, before returning control to the program. As contention winner, the LU avoids having to compete with the partner LU to establish the session, thus potentially saving network traffic. An error in allocating a contention winner session for the conversation is reported on this call.

Sync_level

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not call any services and will not recognize any returned parameters relating to confirmation.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs can call services and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Security_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Security_type specifies the type of access security information that the partner LU uses to verify the identity of the end-user and validate access to the partner program and its resources.

Valid values for this parameter are:

Value**Meaning****100**

Security_none

Specifies to omit access security information on this allocation request.

101

Security_same

Specifies to use the same user ID that is associated with the current program the Allocate service is issued from. The password (if present) is not used; instead, the user ID is indicated as being already verified. If the allocation request that initiated execution of the local program contained no security information, security information is omitted on this allocation request. APPC can retrieve the security information from a number of different places. If the user is authorized and the user specifies a valid User-Token parameter, APPC will use this to obtain the appropriate security information (a user ID and possible profile name). If this is not specified, APPC will send the user ID associated with the current application context environment, if this is available. Otherwise, APPC will send the user ID and possible profile name that is associated with the current executing task, or if unavailable, from the current address space.

102

Security_pgm

Specifies to use the access security information that the local program provides on the call. The local program provides the information by means of the User_ID, Password, and Profile parameters. These values are passed exactly as specified, without folding to uppercase.

Normally, User_ID and Password are required parameters for this Security_type. However, the User_ID parameter can be specified without the Password parameter if, on the local system, the user ID of the issuing address space has been granted surrogate authorization for the specified User_ID. In RACF terms, this requires READ access to the ATBALLC.userid profile (or a generic profile) in the SURROGAT class, where *userid* is the value specified on the User_ID parameter. If surrogate authorization is granted, the user ID specified on the call will be sent and will be indicated as being already verified. For general information on surrogate user IDs, see *z/OS Security Server RACF Security Administrator's Guide*. For specific information about ATBALLC.userid profiles, see *z/OS MVS Planning: APPC/MVS Management*.

Note: If the surrogate authorization is used, the specified User_ID must be a valid MVS user ID. For example, it cannot be longer than 8 characters.

User_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Specifies the user ID. The partner LU uses this value and the password to verify the identity of the end user that initiated the allocation request. The partner LU may use this value for auditing and accounting purposes, and, together with the security profile (if present), to determine which partner programs the local program can access.

When the partner LU is on MVS with RACF protection, the user ID must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Password

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (must be left-justified and padded with blanks if less than 10 bytes)

Specifies the password. The partner LU uses this value and the user ID to verify the identity of the end user that made the allocation request. When the partner LU is on MVS with RACF protection, the password must be 1-8 alphanumeric characters padded with blanks.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Profile

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Profile specifies additional security information that may be used to determine what partner programs the local program may access, and which resources the local program may access. When the partner LU is on MVS with RACF protection, APPC/MVS treats the profile name as a RACF group name for verifying access to partner programs. The profile name must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

User_Token

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 1-255 bytes

User_Token specifies the RACF UTOKEN which identifies the user requesting the Allocate. Only programs running in supervisor state or PSW key 0-7 can specify a User_Token. To not specify a User_Token, pass a field whose first byte contains a hexadecimal zero (X'00').

If a RACF UTOKEN is supplied, APPC/MVS uses it to obtain the appropriate security information only when you specify a Security_Type of Security_Same. In that case, APPC/MVS obtains the user ID and RACF group name from the UTOKEN. This parameter will not be consulted if Security_Type is Security_None or Security_Pgm.

Conversation_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program

immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Note: As of MVS/ESA SP 4.2.2, unauthorized callers can specify a Notify_type of ECB on calls to Allocate. With MVS/ESA SP 4.2, unauthorized callers cannot specify a Notify_type of ECB.

TP_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Allows authorized TPs to designate the transaction program instance with which this conversation should be associated. (See [“Requirements”](#) on page 425 for more information about specific authorization requirements.) Unauthorized TPs must set this parameter to binary zeros, which causes the TP_ID assignment to occur automatically and transparently to the transaction program.

Advanced TPs that run in supervisor state or PSW key 0-7 can select the TP_ID assigned. See the Define_Local_TP callable service description in *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for information on how to create a new TP_ID.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call because nothing is placed in the variables.

An allocation error resulting from the local LU's failure to obtain a session for the conversation is reported on this call. An allocation error resulting from the partner LU's rejection of the allocation request is reported on a subsequent call.

If the Return_control parameter contained a value of When_session_allocated or When_conwinner_allocated, possible values of Return_code are:

Value

Meaning

0

OK

1	Allocate_failure_no_retry
2	Allocate_failure_retry
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_State_Check

If the Return_control parameter contained a value of Immediate, possible values of Return_code are:

Value

Meaning

0	OK
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_State_Check
28	Unsuccessful

For more detailed information about these return codes, refer to [Appendix B, “Explanations of Return Codes for CPI Communications Services,” on page 391.](#)

Restrictions

Transaction programs that call the Allocate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in [z/OS MVS Programming: Authorized Assembler Services Guide](#).

ATBALC2 - Allocate (For MVS/ESA 4.3 through OS/390 Release 7)

Equivalent to:

- LU 6.2 (MC_)Allocate
- CPI Initialize_Conv (CMINIT) and Allocate (CMALLC)

This section describes the Allocate (ATBALC2) TP callable service provided with MVS/ESA Version 4 Release 3. The Allocate service was enhanced for OS/390 Release 8, and renamed ATBALC5 (see

“ATBALC5 - Allocate (For OS/390 Release 8 through z/OS V1R6)” on page 448). The Allocate service was further enhanced for z/OS V1R7, and renamed ATBALC6 (see “Allocate” on page 125). The ATBALC2 and ATBALC5 calls remain valid, but do not contain the enhancements included in ATBALC6.

Note: The ATBALC6 call is the recommended programming interface for this service.

Allocates a session between the local LU and a partner LU, and on that session allocates a basic or mapped conversation between the local program and a partner program. A conversation ID is assigned to the conversation. Call this service before other calls that refer to the conversation.

If the program that issues the allocate call was not started by APPC/MVS in response to an inbound allocate call, and is not associated with an alternative transaction scheduler, the outbound allocate call and ensuing conversation flow through the base LU for the APPC/MVS transaction scheduler. If, in such a case, there is no base LU defined for the APPC/MVS transaction scheduler, APPC/MVS uses the system base LU. If there is no system base LU, APPC/MVS rejects the allocate request. For more information about base LUs and their definition, see *z/OS MVS Planning: APPC/MVS Management*.

Requirements

Authorization:	Supervisor state or problem state, any PSW key, with the following exceptions: <ul style="list-style-type: none"> • When the TP_name specified is an SNA TP name beginning with X'06', the caller must run either in supervisor state, or with PSW key 0-7. • When the TP_id specified is a value other than binary zeros, the caller must run either in supervisor state, or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB mode, with the following exception: task mode only for callers that issue Allocate for a conversation with a synchronization level of Syncpt.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBALC2(
    Conversation_type,
    Sym_dest_name,
    Partner_LU_name,
    Mode_name,
    TP_name_length,
    TP_name,
    Return_control,
    Sync_level,
    Security_type,
    User_ID,
    Password,
    Profile,
    User_Token,
    Conversation_ID,
    Notify_type,
    TP_ID,
    Local_LU_name,
    Return_code
);
```

Figure 66. ATBALC2 - LU 6.2 Allocate

Parameters

Conversation_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the service is invoked.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

Specifies that in this conversation, the TPs will format their data into separate records, with record length and data specified, before sending it.

1

Mapped_conversation

Specifies that in this conversation, the TPs will rely on APPC to format the data that the TPs send.

Sym_dest_name

Supplied parameter

- Type: Character string
- Char Set: 01134
- Length: 8 bytes

Specifies a symbolic name representing the partner LU, the partner TP_name, and the mode name for the session on which the conversation is to be carried. The symbolic destination name must match that of an entry in the side information data set. The appropriate entry in the side information is retrieved and used to initialize the characteristics for the conversation.

If you specify a symbolic destination name, the partner LU name, mode name, and TP name are obtained from the side information. If you also specify values for the Partner_LU_name, Mode_name, or TP_name parameters on the Allocate service, these values override any obtained from the side information.

The symbolic destination name in this field can be from 1 to 8 characters long, with characters from character set 01134. If the symbolic destination name is shorter than eight characters, it must be left-justified in the variable field, and padded on the right with blanks. To not specify a symbolic destination name, set the `sym_dest_name` parameter value to 8 blanks and provide values for the `Partner_LU_name`, `Mode_name`, and `TP_name` parameters.

Partner_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes (must be padded with blanks if less than 17 bytes)

`Partner_LU_name` specifies the name of the LU at which the partner program is located.

The `Partner_LU_name` is any name by which the local LU knows the partner LU for the purposes of allocating a conversation. The local LU transforms this locally known LU name to an LU name used by the network.

The `Partner_LU_name` can be one of the following:

- LU name only (1-8 byte Type A character string).

This string represents the network LU name, which, if unique within the network and interconnected networks, is sufficient for most TP communications.

IBM recommends, however, that you specify either a symbolic destination name (in the `Sym_dest_name` parameter), or the network-qualified LU name, if known. While the network LU name might be unique currently, it might not remain so if the installation increases the number of networks in use. Specifying a symbolic destination name or network-qualified LU name can minimize the need for future network definitions and program changes.

- A VTAM generic resource name.

If the partner LU is a member of a generic resource group, you may specify the 1- to 8-byte generic resource name of the group.

- Combined `network_ID` and network LU name (two 1-8 byte Type A character strings, concatenated by a period: *network_ID.network_LU_name*).

This format is known as a **network-qualified LU name**; each LU in the network and all interconnected networks can be uniquely identified by its network-qualified LU name. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name. If the local LU is not enabled to support network-qualified names, APPC/MVS passes only the network-LU-name portion to VTAM, which might cause an error if the network LU name is not unique across networks.

- A value of 17 blanks:

If you specify a symbolic destination name in `Sym_dest_name` parameter, set `Partner_LU_name` to blanks to use the partner LU name from the side information.

If you do not specify a symbolic destination name, then use a blank `Partner_LU_name` to indicate that the partner program is located at the same LU as the local program (LU=OWN). If the local LU is defined as a member of a VTAM generic resource group, APPC/MVS uses the generic resource name for `Partner_LU_name`.

Mode_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes (must be padded with blanks if less than 8 bytes)

`Mode_name` specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

The mode name value of "SNASVCMG" is reserved for use by APPC/MVS. If a mode name of "SNASVCMG" is specified on the Allocate service, the request is rejected with a return code of parameter_error.

If you specify a symbolic destination name in the sym_dest_name parameter, set mode_name to blanks to obtain the mode_name from the side information.

If you do not specify a sym_dest_name and do not specify a mode name, APPC/MVS uses the default mode name "ATB#MODE".

TP_name_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

TP_name_length specifies the length of data contained in the TP_name parameter.

If you specify a symbolic destination name in the sym_dest_name parameter, set TP_name_length to 0 to use the partner TP name from the side information.

TP_name

Supplied parameter

- Type: Character string
- Char Set: 006409 (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

TP_name specifies the name of the partner program to be connected at the other end of the conversation.

If you specify a symbolic destination name in the sym_dest_name parameter and set the TP_name_length parameter to zero, the TP name is obtained from the side information file.

TP_name can specify the name of any SNA service transaction program except for one whose first character is X'06'; see the authorization requirements in ["Requirements" on page 434](#) for more information about this exception. The names of SNA service transaction programs can contain blank characters. For a list of SNA service transaction programs, see *SNA Transaction Programmer's Reference Document for LU 6.2*.

If the partner TP is to be protected by a RACF security profile in the APPCTP class, the TP_name must consist of Type A characters only.

Return_control

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_control specifies when the local LU is to return control to the local program, in relation to the allocation of a session for the conversation.

Valid values for this parameter are:

Value

Meaning

0

When_session_allocated

Specifies to allocate a session for the conversation before returning control to the program. An error in allocating a session is reported on this call.

1

Immediate

Specifies to allocate a session for the conversation if a session is immediately available, and return control to the program with a return code indicating whether a session is allocated. An error in allocating a session that is immediately available is reported on this call.

100

When_conwinner_allocated

Specifies to allocate a session in which the local LU is the contention winner, before returning control to the program. As contention winner, the LU avoids having to compete with the partner LU to establish the session, thus potentially saving network traffic. An error in allocating a contention winner session for the conversation is reported on this call.

Sync_level

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not call any services and will not recognize any returned parameters relating to confirmation.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs can call services and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Security_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Security_type specifies the type of access security information that the partner LU uses to verify the identity of the end-user and validate access to the partner program and its resources.

Valid values for this parameter are:

Value

Meaning

100

Security_none

Specifies to omit access security information on this allocation request.

101

Security_same

Specifies to use the same user ID that is associated with the current program the Allocate service is issued from. The password (if present) is not used; instead, the user ID is indicated as being already verified. If the allocation request that initiated execution of the local program contained no security information, security information is omitted on this allocation request. APPC can retrieve the security information from a number of different places. If the user is authorized and the user specifies a valid User-Token parameter, APPC will use this to obtain the appropriate security information (a user ID and possible profile name). If this is not specified, APPC will send the user ID associated with the current application context environment, if this is available. Otherwise, APPC will send the user ID and possible profile name that is associated with the current executing task, or if unavailable, from the current address space.

102

Security_pgm

Specifies to use the access security information that the local program provides on the call. The local program provides the information by means of the User_ID, Password, and Profile parameters. These values are passed exactly as specified, without folding to uppercase.

Normally, User_ID and Password are required parameters for this Security_type. However, the User_ID parameter can be specified without the Password parameter if, on the local system, the user ID of the issuing address space has been granted surrogate authorization for the specified User_ID. In RACF terms, this requires READ access to the ATBALLC.userid profile (or a generic profile) in the SURROGAT class, where *userid* is the value specified on the User_ID parameter. If surrogate authorization is granted, the user ID specified on the call will be sent and will be indicated as being already verified. For general information on surrogate user IDs, see *z/OS Security Server RACF Security Administrator's Guide*. For specific information about ATBALLC.userid profiles, see *z/OS MVS Planning: APPC/MVS Management*.

Note: If the surrogate authorization is used, the specified User_ID must be a valid MVS user ID. For example, it cannot be longer than 8 characters.

User_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Specifies the user ID. The partner LU uses this value and the password to verify the identity of the end user that initiated the allocation request. The partner LU may use this value for auditing and accounting purposes, and, together with the security profile (if present), to determine which partner programs the local program can access.

When the partner LU is on MVS with RACF protection, the user ID must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Password

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (must be left-justified and padded with blanks if less than 10 bytes)

Specifies the password. The partner LU uses this value and the user ID to verify the identity of the end user that made the allocation request. When the partner LU is on MVS with RACF protection, the password must be 1-8 alphanumeric characters padded with blanks.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Profile

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Profile specifies additional security information that may be used to determine what partner programs the local program may access, and which resources the local program may access. When the partner LU is on MVS with RACF protection, APPC/MVS treats the profile name as a RACF group name for verifying access to partner programs. The profile name must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

User_Token

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 1-255 bytes

User_Token specifies the RACF UTOKEN which identifies the user requesting the Allocate. Only programs running in supervisor state or PSW key 0-7 can specify a User_Token. To not specify a User_Token, pass a field whose first byte contains a hexadecimal zero (X'00').

If a RACF UTOKEN is supplied, APPC/MVS uses it to obtain the appropriate security information only when you specify a Security_Type of Security_Same. In that case, APPC/MVS obtains the user ID and RACF group name from the UTOKEN. This parameter will not be consulted if Security_Type is Security_None or Security_Pgm.

Conversation_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Note: As of MVS/ESA SP 4.2.2, unauthorized callers can specify a Notify_type of ECB on calls to Allocate. With MVS/ESA SP 4.2, unauthorized callers cannot specify a Notify_type of ECB.

TP_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Allows authorized TPs to designate the transaction program instance with which this conversation should be associated. (See [“Requirements”](#) on page 434 for more information about specific authorization requirements.) Unauthorized TPs must set this parameter to binary zeros, which causes the TP_ID assignment to occur automatically and transparently to the transaction program.

Advanced TPs that run in supervisor state or PSW key 0-7 can select the TP_ID assigned. See the Define_Local_TP callable service description in *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for information on how to create a new TP_ID.

Local_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Local_LU_name specifies the name of the local LU from which the caller's allocate request is to originate. The ability to specify the local LU name allows the caller to associate its outbound conversations with particular LUs. You cannot specify a VTAM generic resource name for the local LU name.

The caller's address space must have access to the named LU. Otherwise, a parameter_error return code is returned. Use [Table 44 on page 442](#) to determine whether you can specify a particular local LU.

Table 44. Local LUs for Which an Address Space Can Allocate

		LU Specified				
		System Base LU, NOSCHED ¹	System Base LU, ASCH ¹	NOSCHED LU	ASCH LU	Scheduler 2 LU
Address Space Doing Allocate	From an Address Space Not Connected to a Scheduler	OK	OK	OK	NO ²	NO ²
	From an Address Space Connected to ASCH	OK	OK	OK	OK	NO ²
	From an Address Space Connected to Scheduler 2	OK	NO ²	OK	NO ²	OK
	From an Address Space Not Connected to a Scheduler with Prohibit Default LU Specified ⁴	NO ³	NO ³	NO ³	NO ³	NO ³

Note:

- Columns 1 and 2 are mutually exclusive.
- The system returns a Parameter_error return code to the caller. If the specified LU is not defined, the system also returns a Product_specific_error return code to the caller.
- The system returns a Product_specific_error return code to the caller.
- For information about how to prohibit the use of a default LU for an address space, see the description of the Set_AS_Attributes service in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

If the caller sets local_LU_name to blanks, the system uses the following hierarchy to select an LU for the conversation:

- The LU associated with the transaction program
- If no LU is associated with the TP, the system uses the base LU for the transaction scheduler associated with the caller's address space.
- If no transaction scheduler is associated with this address space, the system uses the system base LU, which is either:
 - An LU defined with the NOSCHED and BASE parameters, or
 - If a base NOSCHED LU is not defined, the LU defined as the base LU for the APPC/MVS transaction scheduler.
- If no system base LU is defined, the system rejects the Allocate call.

For more information about base LUs and their definitions, see *z/OS MVS Planning: APPC/MVS Management*.

Table 45 on page 443 shows which LU is used by default.

Table 45. Default Local LUs Used If None Are Specified							
Program Calling Allocate Service	Base LUs exist						
	<i>nosched</i>	<i>ASCH</i>	<i>Sched 2</i>	<i>nosched, ASCH</i>	<i>nosched, Sched 2</i>	<i>ASCH, Sched 2</i>	<i>nosched, ASCH, Sched 2</i>
<i>From an Address Space Not Connected to a Scheduler</i>	nosched	asch	NO ¹	nosched	nosched	asch	nosched
<i>From an Address Space Not Connected to a Scheduler but with prohibit default LU specified</i>	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹
<i>From an Address Space Connected to ASCH</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an Address Space Connected to ASCH and with prohibit default LU specified</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an Address Space Connected to Scheduler 2</i>	nosched	NO ¹	Sched 2	nosched	Sched 2	Sched 2	Sched 2
<i>From an Address Space Connected to Scheduler 2 and with prohibit default LU specified</i>	NO ¹	NO ¹	Sched 2	NO ¹	Sched 2	Sched 2	Sched 2
Note: 1. A Product_Specific_Error return code is returned if no base LU exists.							

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call because nothing is placed in the variables.

When APPC/MVS returns an error return code to Allocate, your TP:

- Can use the conversation ID returned on the Conversation_ID parameter as input to the Error_Extract service (which returns detailed information about error return codes)
- Should not examine any other returned parameter associated with the call because no values are placed in the parameters.

An allocation error resulting from the local LU's failure to obtain a session for the conversation is reported on this call. An allocation error resulting from the partner LU's rejection of the allocation request is reported on a subsequent call.

See [“Return Codes” on page 444](#) for descriptions of return codes that can be returned to a caller of Allocate.

Return Codes

If the Return_control parameter contains a value of When_session_allocated or When_conwinner_allocated, possible values of Return_code are:

Decimal Value

Meaning

0	OK
1	Allocate_failure_no_retry
2	Allocate_failure_retry
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check

If the Return_control parameter contains a value of Immediate, possible values of Return_code are:

Decimal Value

Meaning

0	OK
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error
24	Program_parameter_check
25	Program_state_check
28	Unsuccessful

The following table describes *all* of the possible return codes for Allocate:

Table 46. Return Codes for the Allocate Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>
1	<p>Value: Allocate_failure_no_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> • Virtual telecommunications access method (VTAM) could not establish a session with the partner LU. • APPC/MVS could not establish a conversation. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p> <p>If the conversation is not LU=LOCAL, see <i>z/OS Communications Server: SNA Programmer's LU 6.2 Guide</i> for a description of the sense codes included in the message from Error_Extract. If the error persists, or if the conversation is LU=LOCAL, verify that the name specified on the Local_LU_name parameter is correct. If the name is correct, contact the system programmer.</p> <p>System Programmer Response: At the request of the application programmer, ensure that the local LU is defined correctly in the VTAM application (APPL) statement in SYS1.VTAMLST.</p>
2	<p>Value: Allocate_failure_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. The system cannot allocate the conversation because of a condition that might be temporary.</p> <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: Retry the allocate request.</p>
7	<p>Value: Sync_lvl_not_supported_lu</p> <p>Meaning: A TP submitted an Allocate request with a synchronization level that is not supported by the partner LU.</p> <p>System Action: The system returns this return code to the caller of the Allocate call.</p> <p>Application Programmer Response: Ensure that the partner LU supports the receipt of conversations with a synchronization level of syncpt.</p>

Table 46. Return Codes for the Allocate Service (continued)

Return Code	Value, Meaning, and Action
19	<p>Value: Parameter_error</p> <p>Meaning: A local TP called an APPC service. A parameter specified on the call is not valid. The error could be one of the following:</p> <ul style="list-style-type: none"> • The TP name is not 1 to 64 characters long. • Either the SYMDEST name or the TP name length were not specified. • SNASVCMG is specified as mode name. • X'0E' or X'0F' was used as the first character of a TP name. • X'06' was used as the first character of a TP name by a caller that was not running either in supervisor state or with PSW key 0-7. • An SNA service TP name is used with a mapped conversation verb. • The partner LU name was not valid. • The mode name was not valid. • The local LU name specified is either undefined or not allowed; for example, the TP might have specified a VTAM generic resource name, which is valid only for partner LU names. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 46. Return Codes for the Allocate Service (continued)	
Return Code	Value, Meaning, and Action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The local TP called an APPC service. One of the following errors occurred in one or more parameters specified on the call:</p> <ul style="list-style-type: none"> • An unauthorized caller passed a non-zero TP_ID. • For a Security_type of Security_pgm, both the user ID and password were not specified. • For a Security_type of Security_pgm, a user ID was specified with a blank password, or a password was specified with a blank user ID. • The SYMDEST name was not found in the side information. • The specified TP_ID is not associated with the address space. • An unauthorized caller specified a UTOKEN that was non-zero. • The specified local LU does not support protected conversations (conversations with a synchronization level of syncpt). <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: For a conversation with sync_level set to SYNCPT, the conversation's context (unit of work) is in the Backout-Required condition. New protected conversations cannot be allocated for a context in this condition.</p> <p>System Action: The conversation allocation request fails. A new conversation is not allocated.</p> <p>Application Programmer Response: Backout the current unit of recovery associated with the transaction program's context.</p>
28	<p>Value: Unsuccessful</p> <p>Meaning: The request specified an allocate_type of <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> • APPC/MVS could not establish a session with the partner LU • Virtual telecommunications access method (VTAM) could not establish a conversation. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

For more detailed information about these return codes, refer to [Appendix B, “Explanations of Return Codes for CPI Communications Services,”](#) on page 391.

Restrictions

Transaction programs that call the Allocate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT

FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

ATBALC5 - Allocate (For OS/390 Release 8 through z/OS V1R6)

Equivalent to:

- LU 6.2 (MC_)Allocate
- CPI Initialize_Conv (CMINIT) and Allocate (CMALLC)

This section describes the Allocate (ATBALC5) TP callable service provided with OS/390 Release 8. The Allocate service was enhanced for z/OS V1R7, and renamed ATBALC6 (see [“Allocate” on page 125](#)). The ATBALC5 call remains valid, but does not contain the enhancements included in ATBALC6.

Note: The ATBALC6 call is the recommended programming interface for this service.

Allocates a session between the local LU and a partner LU, and on that session allocates a basic or mapped conversation between the local program and a partner program. A conversation ID is assigned to the conversation. Call this service before other calls that refer to the conversation.

If the program that issues the allocate call was not started by APPC/MVS in response to an inbound allocate call, and is not associated with an alternative transaction scheduler, the outbound allocate call and ensuing conversation flow through the base LU for the APPC/MVS transaction scheduler. If, in such a case, there is no base LU defined for the APPC/MVS transaction scheduler, APPC/MVS uses the system base LU. If there is no system base LU, APPC/MVS rejects the allocate request. For more information about base LUs and their definition, see *z/OS MVS Planning: APPC/MVS Management*.

Requirements

Authorization:	Supervisor state or problem state, any PSW key, with the following exceptions: <ul style="list-style-type: none"> • When the TP_name specified is an SNA TP name beginning with X'06', the caller must run either in supervisor state, or with PSW key 0-7. • When the TP_id specified is a value other than binary zeros, the caller must run either in supervisor state, or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB mode, with the following exception: task mode only for callers that issue Allocate for a conversation with a synchronization level of Syncpt.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31-bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	Unlocked.
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBALC5(
    Conversation_type,
    Sym_dest_name,
    Partner_LU_name,
    Mode_name,
    TP_name_length,
    TP_name,
    Return_control,
    Sync_level,
    Security_type,
    User_ID,
    Password,
    Profile,
    User_Token,
    Conversation_ID,
    Notify_type,
    TP_ID,
    Local_LU_name,
    Timeout_value,
    Return_code
);
```

Figure 67. ATBALC5 - LU 6.2 Allocate

Parameters

Conversation_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Conversation_type specifies the type of conversation on which the service is invoked.

Valid values for this parameter are:

Value

Meaning

0

Basic_conversation

Specifies that in this conversation, the TPs will format their data into separate records, with record length and data specified, before sending it.

1

Mapped_conversation

Specifies that in this conversation, the TPs will rely on APPC to format the data that the TPs send.

Sym_dest_name

Supplied parameter

- Type: Character string
- Char Set: 01134
- Length: 8 bytes

Specifies a symbolic name representing the partner LU, the partner TP_name, and the mode name for the session on which the conversation is to be carried. The symbolic destination name must match that of an entry in the side information data set. The appropriate entry in the side information is retrieved and used to initialize the characteristics for the conversation.

If you specify a symbolic destination name, the partner LU name, mode name, and TP name are obtained from the side information. If you also specify values for the Partner_LU_name, Mode_name, or TP_name parameters on the Allocate service, these values override any obtained from the side information.

The symbolic destination name in this field can be from 1 to 8 characters long, with characters from character set 01134. If the symbolic destination name is shorter than eight characters, it must be left-justified in the variable field, and padded on the right with blanks. To not specify a symbolic destination name, set the sym_dest_name parameter value to 8 blanks and provide values for the Partner_LU_name, Mode_name, and TP_name parameters.

Partner_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes (must be padded with blanks if less than 17 bytes)

Partner_LU_name specifies the name of the LU at which the partner program is located.

The Partner_LU_name is any name by which the local LU knows the partner LU for the purposes of allocating a conversation. The local LU transforms this locally known LU name to an LU name used by the network.

The Partner_LU_name can be one of the following:

- LU name only (1-8 byte Type A character string).

This string represents the network LU name, which, if unique within the network and interconnected networks, is sufficient for most TP communications.

IBM recommends, however, that you specify either a symbolic destination name (in the Sym_dest_name parameter), or the network-qualified LU name, if known. While the network LU name might be unique currently, it might not remain so if the installation increases the number of networks in use. Specifying a symbolic destination name or network-qualified LU name can minimize the need for future network definitions and program changes.

- A VTAM generic resource name.

If the partner LU is a member of a generic resource group, you may specify the 1- to 8-byte generic resource name of the group.

- Combined network_ID and network LU name (two 1-8 byte Type A character strings, concatenated by a period: *network_ID.network_LU_name*).

This format is known as a **network-qualified LU name**; each LU in the network and all interconnected networks can be uniquely identified by its network-qualified LU name. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name. If the local LU is not enabled to support network-qualified names, APPC/MVS passes only the network-LU-name portion to VTAM, which might cause an error if the network LU name is not unique across networks.

- A value of 17 blanks:

If you specify a symbolic destination name in Sym_dest_name parameter, set Partner_LU_name to blanks to use the partner LU name from the side information.

If you do not specify a symbolic destination name, then use a blank Partner_LU_name to indicate that the partner program is located at the same LU as the local program (LU=OWN). If the local LU is defined as a member of a VTAM generic resource group, APPC/MVS uses the generic resource name for Partner_LU_name.

Mode_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes (must be padded with blanks if less than 8 bytes)

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

The mode name value of "SNASVCMG" is reserved for use by APPC/MVS. If a mode name of "SNASVCMG" is specified on the Allocate service, the request is rejected with a return code of parameter_error.

If you specify a symbolic destination name in the sym_dest_name parameter, set mode_name to blanks to obtain the mode_name from the side information.

If you do not specify a sym_dest_name and do not specify a mode name, APPC/MVS uses the default mode name "ATB#MODE".

TP_name_length

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

TP_name_length specifies the length of data contained in the TP_name parameter.

If you specify a symbolic destination name in the sym_dest_name parameter, set TP_name_length to 0 to use the partner TP name from the side information.

TP_name

Supplied parameter

- Type: Character string
- Char Set: 006409 (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

TP_name specifies the name of the partner program to be connected at the other end of the conversation.

If you specify a symbolic destination name in the sym_dest_name parameter and set the TP_name_length parameter to zero, the TP name is obtained from the side information file.

TP_name can specify the name of any SNA service transaction program except for one whose first character is X'06'; see the authorization requirements in ["Requirements" on page 448](#) for more information about this exception. The names of SNA service transaction programs can contain blank characters. For a list of SNA service transaction programs, see *SNA Transaction Programmer's Reference Document for LU 6.2*.

If the partner TP is to be protected by a RACF security profile in the APPCTP class, the TP_name must consist of Type A characters only.

Return_control

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_control specifies when the local LU is to return control to the local program, in relation to the allocation of a session for the conversation.

Valid values for this parameter are:

Value

Meaning

0

When_session_allocated

Specifies to allocate a session for the conversation before returning control to the program. An error in allocating a session is reported on this call.

1

Immediate

Specifies to allocate a session for the conversation if a session is immediately available, and return control to the program with a return code indicating whether a session is allocated. An error in allocating a session that is immediately available is reported on this call.

100

When_conwinner_allocated

Specifies to allocate a session in which the local LU is the contention winner, before returning control to the program. As contention winner, the LU avoids having to compete with the partner LU to establish the session, thus potentially saving network traffic. An error in allocating a contention winner session for the conversation is reported on this call.

Sync_level

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not call any services and will not recognize any returned parameters relating to confirmation.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs can call services and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Security_type

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Security_type specifies the type of access security information that the partner LU uses to verify the identity of the end-user and validate access to the partner program and its resources.

Valid values for this parameter are:

Value

Meaning

100

Security_none

Specifies to omit access security information on this allocation request.

101

Security_same

Specifies to use the same user ID that is associated with the current program the Allocate service is issued from. The password (if present) is not used; instead, the user ID is indicated as being already verified. If the allocation request that initiated execution of the local program contained no security information, security information is omitted on this allocation request. APPC can retrieve the security information from a number of different places. If the user is authorized and the user specifies a valid User-Token parameter, APPC will use this to obtain the appropriate security information (a user ID and possible profile name). If this is not specified, APPC will send the user ID associated with the current application context environment, if this is available. Otherwise, APPC will send the user ID and possible profile name that is associated with the current executing task, or if unavailable, from the current address space.

102

Security_pgm

Specifies to use the access security information that the local program provides on the call. The local program provides the information by means of the User_ID, Password, and Profile parameters. These values are passed exactly as specified, without folding to uppercase.

Normally, User_ID and Password are required parameters for this Security_type. However, the User_ID parameter can be specified without the Password parameter if, on the local system, the user ID of the issuing address space has been granted surrogate authorization for the specified User_ID. In RACF terms, this requires READ access to the ATBALLC.userid profile (or a generic profile) in the SURROGAT class, where *userid* is the value specified on the User_ID parameter. If surrogate authorization is granted, the user ID specified on the call will be sent and will be indicated as being already verified. For general information on surrogate user IDs, see *z/OS Security Server RACF Security Administrator's Guide*. For specific information about ATBALLC.userid profiles, see *z/OS MVS Planning: APPC/MVS Management*.

Note: If surrogate authorization is used, the specified User_ID must be a valid MVS user ID. For example, it cannot be longer than 8 characters.

User_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Specifies the user ID. The partner LU uses this value and the password to verify the identity of the end user that initiated the allocation request. The partner LU may use this value for auditing and accounting purposes, and, together with the security profile (if present), to determine which partner programs the local program can access.

When the partner LU is on MVS with RACF protection, the user ID must be 1-8 alphanumeric characters.

This parameter is significant only when the Security_type parameter contains a value of Pgm. Otherwise, this parameter has no meaning and is ignored.

Password

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (must be left-justified and padded with blanks if less than 10 bytes)

Specifies the password. The partner LU uses this value and the user ID to verify the identity of the end user that made the allocation request. When the partner LU is on MVS with RACF protection, the password must be 1-8 alphanumeric characters padded with blanks.

This parameter is significant only when the `Security_type` parameter contains a value of `Pgm`. Otherwise, this parameter has no meaning and is ignored.

Profile

Supplied parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes

Profile specifies additional security information that may be used to determine what partner programs the local program may access, and which resources the local program may access. When the partner LU is on MVS with RACF protection, APPC/MVS treats the profile name as a RACF group name for verifying access to partner programs. The profile name must be 1-8 alphanumeric characters.

This parameter is significant only when the `Security_type` parameter contains a value of `Pgm`. Otherwise, this parameter has no meaning and is ignored.

User_Token

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 1-255 bytes

`User_Token` specifies the RACF UTOKEN which identifies the user requesting the Allocate. Only programs running in supervisor state or PSW key 0-7 can specify a `User_Token`. To not specify a `User_Token`, pass a field whose first byte contains a hexadecimal zero (X'00').

If a RACF UTOKEN is supplied, APPC/MVS uses it to obtain the appropriate security information only when you specify a `Security_Type` of `Security_Same`. In that case, APPC/MVS obtains the user ID and RACF group name from the UTOKEN. This parameter will not be consulted if `Security_Type` is `Security_None` or `Security_Pgm`.

Conversation_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

`Conversation_id`, sometimes called the resource identifier, identifies a conversation to the system.

Notify_type

Supplied parameter

- Type: Structure
- Char Set: N/A
- Length: 4-8 bytes

Specifies the type of processing and notification (synchronous or asynchronous) requested for this service. Programs can request asynchronous processing, which returns control to the program immediately and later notifies the program by ECB when the service is complete. The possible types are:

- None

No notification is requested. The service is performed synchronously, and control is returned to the caller when processing is complete. All returned parameters are set on return to the caller. To specify no notification, set the parameter value to a four-byte structure containing binary zeros.

- ECB

Programs can request asynchronous processing by specifying an ECB to be posted when processing completes. To specify an ECB, set the parameter to an eight-byte structure containing a fullword binary one (X'00000001') followed by the address of a fullword area to be used as the ECB. The ECB must reside in the home address space.

When you specify an ECB, control is returned before processing is complete, with only the return code set. If the asynchronous request was accepted, the return code is set to 0 to indicate that the service is being processed asynchronously. Other returned parameters are filled in during asynchronous processing, and the specified ECB is posted when all returned parameters are set. The completion code field in the ECB contains the return code for the service.

Note: As of MVS/ESA SP 4.2.2, unauthorized callers can specify a Notify_type of ECB on calls to Allocate. With MVS/ESA SP 4.2, unauthorized callers cannot specify a Notify_type of ECB.

TP_ID

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Allows authorized TPs to designate the transaction program instance with which this conversation should be associated. (See “Requirements” on page 448 for more information about specific authorization requirements.) Unauthorized TPs must set this parameter to binary zeros, which causes the TP_ID assignment to occur automatically and transparently to the transaction program.

Advanced TPs that run in supervisor state or PSW key 0-7 can select the TP_ID assigned. See the Define_Local_TP callable service description in *z/OS MVS System Messages, Vol 3 (ASB-BPX)* for information on how to create a new TP_ID.

Local_LU_name

Supplied parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Local_LU_name specifies the name of the local LU from which the caller's allocate request is to originate. The ability to specify the local LU name allows the caller to associate its outbound conversations with particular LUs. You cannot specify a VTAM generic resource name for the local LU name.

The caller's address space must have access to the named LU. Otherwise, a parameter_error return code is returned. Use [Table 47 on page 455](#) to determine whether you can specify a particular local LU.

Table 47. Local LUs for Which an Address Space Can Allocate					
Address space doing allocate	LU specified				
	System base LU, NOSCHED ¹	System base LU, ASCH ¹	NOSCHED LU	ASCH LU	Scheduler 2 LU
From an address space not connected to a scheduler	OK	OK	OK	NO ²	NO ²

Table 47. Local LUs for Which an Address Space Can Allocate (continued)

Address space doing allocate	LU specified				
	System base LU, NOSCHED ¹	System base LU, ASCH ¹	NOSCHED LU	ASCH LU	Scheduler 2 LU
From an address space connected to ASCH	OK	OK	OK	OK	NO ²
From an address space connected to Scheduler 2	OK	NO ²	OK	NO ²	OK
From an address space not connected to a scheduler with Prohibit Default LU specified ⁴	NO ³	NO ³	NO ³	NO ³	NO ³

Notes:

- Columns 2 (System base LU, NOSCHED) and 3 (System base LU, ASCH) are mutually exclusive.
- The system returns a Parameter_error return code to the caller. If the specified LU is not defined, the system also returns a Product_specific_error return code to the caller.
- The system returns a Product_specific_error return code to the caller.
- For information about how to prohibit the use of a default LU for an address space, see the description of the Set_AS_Attributes service in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

If the caller sets local_LU_name to blanks, the system uses the following hierarchy to select an LU for the conversation:

- The LU associated with the transaction program
- If no LU is associated with the TP, the system uses the base LU for the transaction scheduler associated with the caller's address space.
- If no transaction scheduler is associated with this address space, the system uses the system base LU, which is either:
 - An LU defined with the NOSCHED and BASE parameters, or
 - If a base NOSCHED LU is not defined, the LU defined as the base LU for the APPC/MVS transaction scheduler.
- If no system base LU is defined, the system rejects the Allocate call.

For more information about base LUs and their definitions, see *z/OS MVS Planning: APPC/MVS Management*.

Table 48 on page 457 shows which LU is used by default.

Table 48. Default Local LUs Used If None Are Specified							
Program calling allocate service	Base LUs exist						
	<i>nosched</i>	<i>ASCH</i>	<i>Sched 2</i>	<i>nosched, ASCH</i>	<i>nosched, Sched 2</i>	<i>ASCH, Sched 2</i>	<i>nosched, ASCH, Sched 2</i>
<i>From an address space not connected to a scheduler</i>	nosched	asch	NO ¹	nosched	nosched	asch	nosched
<i>From an address space not connected to a scheduler but with Prohibit Default LU specified</i>	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹	NO ¹
<i>From an address space connected to ASCH</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an address space connected to ASCH and with Prohibit Default LU specified</i>	N/A	N/A	N/A	N/A	N/A	N/A	N/A
<i>From an address space connected to Scheduler 2</i>	nosched	NO ¹	Sched 2	nosched	Sched 2	Sched 2	Sched 2
<i>From an address space connected to Scheduler 2 and with Prohibit Default LU specified</i>	NO ¹	NO ¹	Sched 2	NO ¹	Sched 2	Sched 2	Sched 2
Note: 1. A Product_Specific_Error return code is returned if no base LU exists.							

Timeout_value

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Value range: 0-1440 (decimal)

Sets a time limit in minutes that an allocate call and subsequent APPC/MVS TP conversation calls will wait for VTAM APPCCMD requests to complete. For more information, see [“Setting a Timeout Value for Potential Network Delays” on page 55.](#)

If the time limit is reached before the VTAM APPCCMD request completes and returns control to APPC/MVS, the conversation will be terminated by APPC/MVS and the caller of the conversation

callable service will regain control. If the conversation call was issued with a Notify_Type=ECB (asynchronous processing), the specified ECB will be posted when the time limit is reached.

To alter the timeout_value set on the Allocate service, use the Set_Timeout_Value service.

The maximum supported Timeout_Value is 1440 minutes (24 hours). A Timeout_Value of 0 and any other positive integer (<= 1440) is valid. When a Timeout_Value of zero is specified, the allocate call and any subsequent APPC/MVS TP conversation calls will not be timed. You can activate the time-out feature later by invoking the Set_Timeout_Value conversation callable service and specifying a non-zero Timeout_Value.

When a non-zero Timeout_Value is specified and a VTAM APPCCMD request issued during allocate processing does not complete within the time-out period, the conversation allocation will fail and APPC/MVS will return control to the application with a Product_Specific_Error return code.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call because nothing is placed in the variables.

When APPC/MVS returns an error return code to Allocate, your TP:

- Can use the conversation ID returned on the Conversation_ID parameter as input to the Error_Extract service (which returns detailed information about error return codes)
- Should not examine any other returned parameter associated with the call because no values are placed in the parameters.

An allocation error resulting from the local LU's failure to obtain a session for the conversation is reported on this call. An allocation error resulting from the partner LU's rejection of the allocation request is reported on a subsequent call.

See ["Return Codes" on page 458](#) for descriptions of return codes that can be returned to a caller of Allocate.

Return Codes

If the Return_control parameter contains a value of When_session_allocated or When_conwinner_allocated, possible values of Return_code are:

Decimal Value

Meaning

0	OK
1	Allocate_failure_no_retry
2	Allocate_failure_retry
7	Sync_lvl_not_supported_lu
19	Parameter_error
20	Product_specific_error

24

Program_parameter_check

25

Program_state_check

If the Return_control parameter contains a value of Immediate, possible values of Return_code are:

Decimal Value**Meaning****0**

OK

7

Sync_lvl_not_supported_lu

19

Parameter_error

20

Product_specific_error

24

Program_parameter_check

25

Program_state_check

28

Unsuccessful

The following table describes *all* of the possible return codes for Allocate:

<i>Table 49. Return Codes for the Allocate Service</i>	
Return code	Value, meaning, and action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: If the call specified a Notify_type of ECB, APPC/MVS posts the ECB specified on the Notify_type parameter when APPC/MVS finishes processing the call asynchronously.</p> <p>Application Programmer Response: None required.</p>

Table 49. Return Codes for the Allocate Service (continued)	
Return code	Value, meaning, and action
1	<p>Value: Allocate_failure_no_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> Virtual telecommunications access method (VTAM) could not establish a session with the partner LU. APPC/MVS could not establish a conversation. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p> <p>If the conversation is not LU=LOCAL, see <i>z/OS Communications Server: SNA Programmer's LU 6.2 Guide</i> for a description of the sense codes included in the message from Error_Extract. If the error persists, or if the conversation is LU=LOCAL, verify that the name specified on the Local_LU_name parameter is correct. If the name is correct, contact the system programmer.</p> <p>System Programmer Response: At the request of the application programmer, ensure that the local LU is defined correctly in the VTAM application (APPL) statement in SYS1.VTAMLST.</p>
2	<p>Value: Allocate_failure_retry</p> <p>Meaning: A TP submitted an allocate request. The request specified a value on the Return_control parameter that was other than <i>Immediate</i>. The system cannot allocate the conversation because of a condition that might be temporary.</p> <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: Retry the allocate request.</p>
7	<p>Value: Sync_lvl_not_supported_lu</p> <p>Meaning: A TP submitted an Allocate request with a synchronization level that is not supported by the partner LU.</p> <p>System Action: The system returns this return code to the caller of the Allocate call.</p> <p>Application Programmer Response: Ensure that the partner LU supports the receipt of conversations with a synchronization level of syncpt.</p>

Table 49. Return Codes for the Allocate Service (continued)

Return code	Value, meaning, and action
19	<p>Value: Parameter_error</p> <p>Meaning: A local TP called an APPC service. A parameter specified on the call is not valid. The error could be one of the following:</p> <ul style="list-style-type: none"> • The TP name is not 1 to 64 characters long. • Either the SYMDEST name or the TP name length were not specified. • SNASVCMG is specified as mode name. • X'0E' or X'0F' was used as the first character of a TP name. • X'06' was used as the first character of a TP name by a caller that was not running either in supervisor state or with PSW key 0-7. • An SNA service TP name is used with a mapped conversation verb. • The partner LU name was not valid. • The mode name was not valid. • The local LU name specified is either undefined or not allowed; for example, the TP might have specified a VTAM generic resource name, which is valid only for partner LU names. <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 49. Return Codes for the Allocate Service (continued)	
Return code	Value, meaning, and action
24	<p>Value: Program_parameter_check</p> <p>Meaning: The local TP called an APPC service. One of the following errors occurred in one or more parameters specified on the call:</p> <ul style="list-style-type: none"> • An unauthorized caller passed a non-zero TP_ID. • For a Security_type of Security_pgm, both the user ID and password were not specified. • For a Security_type of Security_pgm, a user ID was specified with a blank password, or a password was specified with a blank user ID. • The SYMDEST name was not found in the side information. • The specified TP_ID is not associated with the address space. • An unauthorized caller specified a UTOKEN that was non-zero. • The specified local LU does not support protected conversations (conversations with a synchronization level of syncpt). • The specified Timeout_value is not valid <p>System Action: The system returns this return code to the caller of Allocate.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>
25	<p>Value: Program_state_check</p> <p>Meaning: For a conversation with sync_level set to SYNCPT, the conversation's context (unit of work) is in the Backout-Required condition. New protected conversations cannot be allocated for a context in this condition.</p> <p>System Action: The conversation allocation request fails. A new conversation is not allocated.</p> <p>Application Programmer Response: Backout the current unit of recovery associated with the transaction program's context.</p>
28	<p>Value: Unsuccessful</p> <p>Meaning: The request specified an allocate_type of <i>Immediate</i>. One of the following occurred:</p> <ul style="list-style-type: none"> • APPC/MVS could not establish a session with the partner LU • Virtual telecommunications access method (VTAM) could not establish a conversation. <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

For more detailed information about these return codes, refer to [Appendix B, “Explanations of Return Codes for CPI Communications Services,” on page 391.](#)

Restrictions

Transaction programs that call the Allocate service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT

FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

ATBCMCTU - Cleanup_TP (Unauthorized, for MVS/ESA 4.2)

This section describes the Cleanup_TP (ATBCMCTU) advanced callable service provided with MVS/ESA Version 4 Release 2. The Cleanup_TP service was enhanced for MVS/ESA Version 4 Release 2.2, and renamed ATBCUC1 (see “Cleanup_TP” on page 263). The ATBCMCTU call remains valid in MVS/ESA 4.2.2, but does not contain the enhancements included in ATBCUC1. However, if this call is issued against a TP that has active protected conversations, ATBCMCTU will have the same effect on the protected conversations as ATBCUC1 does.

Note: The ATBCUC1 call is the recommended programming interface for this service.

You can call ATBCMCTU from an unauthorized program to request that the APPC component clean up all conversation resources associated with a transaction program instance that is running in the caller's address space. Conversation resources include network resources, control blocks, and buffers which are used by the APPC component to manage the transaction program instance and its conversations.

ATBCMCTU is an unauthorized version of the ATBCMTP service described in *z/OS MVS System Messages, Vol 3 (ASB-BPX)*.

The primary use for ATBCMCTU is to clean up conversation resources left after testing a TP with the Register_Test and Accept_Test services.

The specified TP_ID is deleted from the system as a result of this call, but cleanup processing occurs asynchronously. Conversations with active APPC requests are not immediately deallocated. After the partner TP responds, APPC/MVS returns a deallocate condition and deallocates the conversation.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBCMCTU (TP_ID,
               Condition,
               Return_Code
               );
```

Figure 68. ATBCMCTU - Cleanup_TP (Unauthorized Version)

Parameters

TP_ID

Supplied parameter

- Type: Character string

- Char Set: No restriction
- Length: 8 bytes

Specifies the transaction program instance which is to be cleaned up. All conversations owned by this transaction program instance are to be deallocated.

Condition

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Specifies the deallocation condition that has occurred. This field is used to determine the type of deallocate and sense code that is issued by the APPC component to the partner transaction program.

Valid values for this parameter are:

Value

Meaning

0

Normal

Specifies that the transaction program completed normally, even though it may have left active conversations. The APPC component deallocates all conversations in a proper state for normal deallocation with DEALLOCATE TYPE(SYNC_LEVEL). All conversations not in the proper state for a normal deallocation are deallocated with TYPE(ABEND_SVC).

1

System

Specifies that the transaction program terminated abnormally. All active conversations are deallocated with TYPE(ABEND_SVC).

Note: If the value is not one of the values listed above, 0 (Normal) is used as the default.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

ATBCMCTU may return one of the following values in the return code parameter:

Decimal Value

Meaning

0

Request accepted. All conversations owned by the transaction program instance will be cleaned up asynchronously.

4

No conversations exist to be cleaned up.

8

The TP_ID parameter specified a nonexistent transaction program instance or a transaction program not in the caller's home address space.

32

The requested service is not supported in the caller's environment. For example, this return code will be given if the caller invokes any of the transaction scheduler services while holding a system lock.

44

APPC/MVS is not active.

Restrictions

- Transaction programs that call the Cleanup_TP service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.
- Regardless of the condition parameter value specified for this service, APPC/MVS cleans up protected conversations differently, depending on whether a syncpoint operation is in progress. When a syncpoint operation **is** in progress for the current UR for the context with which the protected conversation is associated, APPC/MVS does not immediately deallocate the conversation. The syncpoint operation is allowed to complete. As part of the syncpoint processing, the protected conversation might be deallocated, in which case no further cleanup is required for that conversation.

If the conversation was not deallocated, however, cleanup processing proceeds in the same manner as it does when a syncpoint operation **is not** in progress at the time the Cleanup service is issued:

- The protected conversation is deallocated with TYPE(ABEND_SVC).
- The current UR is put into backout-required state.
- If the protected conversation is an inbound conversation, the logical unit of work ID (LUWID) for the next UR is reset.
- The current UR and subsequent units of recovery for the context will not include the protected conversation being cleaned up by this service.

ATBGTA2 - Get_Attributes (For MVS/ESA 4.3 through z/OS V1R6)

Related to:

- LU 6.2 (MC_)Get_Attributes
- CPI Communications Extract_Conv_State (CMECS), Extract_Mode_Name (CMEMN), Extract_Part_LU_Name (CMEPLN), Extract_Sync_Level (CMESL)

This section describes the Get_Attributes callable service provided with MVS/ESA 4.3. The Get_Attributes service was enhanced for z/OS V1R7, and renamed ATBGTA6 (see [“Get_Attributes” on page 165](#)). The ATBGTA2 call remains valid in z/OS V1R6, but does not contain the enhancements included in ATBGTA6.

Note: The ATBGTA6 call is the recommended programming interface for this service.

Determines certain attributes of the conversation specified by the Conversation_ID parameter. This service is most useful when issued for an inbound conversation; the returned parameters provide important information about the attributes with which the conversation was allocated. Some of the data returned by Get_Attributes can be obtained only through this service. However, several fields provide information that is also available from other APPC/MVS services. For example, you can determine the conversation_type returned by Get_Attributes by calling the Get_Type service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts

Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGTA2(
    Conversation_id,
    Partner_LU_name,
    Mode_name,
    Sync_level,
    Conversation_correlator,
    LUW_id,
    TP_name_length,
    TP_name,
    Local_LU_name,
    Conversation_type,
    User_id,
    Profile,
    User_token,
    Conversation_state,
    Return_code
);
```

Figure 69. ATBGTA2 - LU 6.2 Get Attributes

Parameters

Conversation_id

Supplied parameter

- Type: Character string
- Char Set: No restriction
- Length: 8 bytes

Conversation_id, sometimes called the resource identifier, identifies a conversation to the system.

Partner_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes

Partner_LU_name specifies the name of the LU at which the partner program is located.

The Partner_LU_name can be one of the following:

- A VTAM generic resource name

If the partner LU is a member of a generic resource group, the Partner_LU_name might be the 1- to 8-byte generic resource name of the group.

- The network-qualified name of the partner logical unit.

The network-qualified name consists of two Type A character strings that represent the network ID and network LU name, respectively. Both strings are between 1 and 8 bytes in length, concatenated together by a period: *network_ID.network_LU_name*. The network-LU-name portion may be a VTAM generic resource name, or a specific LU name.

- A Type A character string that is 1 to 8 bytes in length. This string represents the network LU name.

Mode_name

Returned parameter

- Type: Character string

- Char Set: Type A
- Length: 8 bytes

Mode_name specifies the mode name designating the network properties for the session to be allocated for the conversation. The network properties include, for example, the class of service to be used.

If Mode_name is less than 8 bytes in length, it is padded on the right with blanks.

Sync_level

Returned parameter

- Type: Integer
- Length: 32 bits

Sync_level specifies the synchronization level that the local and partner programs can use on this conversation.

Valid values for this parameter are:

Value

Meaning

0

None

Specifies that the programs will not perform confirmation processing on this conversation. The programs will not issue any protocol boundary calls and will not recognize any returned parameters relating to synchronization functions.

1

Confirm

Specifies that the programs can perform confirmation processing on this conversation. The programs may issue protocol boundary calls and will recognize returned parameters relating to confirmation.

2

Syncpt

Specifies that the programs can perform sync point processing on this conversation. The programs can call services and will recognize returned parameters relating to sync point processing.

Conversation_correlator

Returned parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_correlator specifies further qualification of the LU work identifier (LUW_id) and helps restore protected resources to a consistent state following the failure of an LU, session, or conversation.

If there is no conversation correlator for the conversation, this field contains binary zeroes.

LUW_id

Returned parameter

- Type: Structure
- Char set: N/A
- Length: 26 bytes

LUW_id contains the logical unit of work (LUW) identifier. The LUW identifier is used by some logical units for accounting purposes. If the value returned on the Sync_level parameter is syncpt,

a protected LUW_id is returned in this parameter. If no LUW identifier is present, this field contains binary zeroes.

TP_name_length

Returned parameter

- Type: Integer
- Length: 32 bits

TP_name_length contains the length of the data in the TP_name parameter. If the conversation_id parameter specifies an outbound conversation, this field is set to zero.

TP_name

Returned parameter

- Type: Character string
- Char Set: 00640 or Type A (Type A if the partner TP is protected by RACF)
- Length: 1-64 bytes

When the conversation_id parameter specifies an inbound conversation, TP_name contains the name of the local TP for this conversation. If you called the Register_For_Allocates service to become a server of inbound allocate requests, this parameter contains the TP name specified in the FMH-5 that contained the request. When the conversation_id parameter specifies an outbound conversation, this field is not set.

Local_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 8 bytes

Local_LU_name specifies the name of the local LU from which the conversation is initiated.

Conversation_type

Returned parameter

- Type: Integer
- Length: 32 bits

Conversation_type specifies how the data sent on this conversation is to be formatted.

Valid values for this parameter are:

Value
Meaning

0

Basic_conversation

Specifies that, in this conversation, the calling program and its partner will format their data into separate logical records before sending it. Each record begins with a 2-byte length field (LL) that specifies the amount of data in the record.

1

Mapped_conversation

Specifies that, in this conversation, the calling program and its partner will rely on APPC to format the data they send.

User_id

Returned parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)

- Length: 10 bytes (left-justified if the user ID is less than 10 bytes)

If the Conversation_id parameter specifies an inbound conversation, User_id returns the user ID associated with the inbound allocate request. If the Conversation_id parameter specifies an outbound conversation, this field contains blanks.

Profile

Returned parameter

- Type: Character string
- Char Set: No restriction (Type A if APPC/MVS manages the partner LU)
- Length: 10 bytes (left-justified if the profile name is less than 10 bytes)

If the Conversation_id parameter specifies an inbound conversation, profile contains the RACF group name associated with the inbound allocate request. When the Conversation_id parameter specifies an outbound conversation, this field contains blanks.

User_token

Returned parameter

- Type: Structure
- Char Set: N/A
- Length: 80 bytes

If the Conversation_id parameter specifies an inbound conversation, User_token contains the RACF UTOKEN that identifies the user associated with the inbound allocate. This token is encrypted.

If the conversation_id parameter specifies an outbound conversation, this field is blanks. No UTOKEN is returned for an outbound conversation.

Conversation_state

Returned parameter

- Type: Integer
- Length: 32 bits

Conversation_state specifies the current state of the conversation, which is one of the following:

Value

Conversation State

2

Initialize

3

Send

4

Receive

5

Send-Pending

6

Confirm

7

Confirm-Send

8

Confirm-Deallocate

9

Defer-Receive

10

Defer-Deallocate

11

Sync-Point

12

Sync-Point-Send

13

Sync-Point-Deallocate

For descriptions, see [“APPC/MVS TP Conversation States”](#) on page 45.

Return_code

Returned parameter

- Type: Integer
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See [“Return Codes”](#) on page 470 for descriptions of return codes that can be returned to a caller of Get_Attributes.

Return Codes

Valid return code values for the Return_code parameter are:

Table 50. Return Codes for the Get_Attributes Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_parameter_check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to the caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Table 50. Return Codes for the Get_Attributes Service (continued)

Return Code	Value, Meaning, and Action
100	<p>Value: Take_backout</p> <p>Meaning: This value is returned only when all of the following conditions are true:</p> <ul style="list-style-type: none"> • The Sync_level is set to syncpt. • The conversation is not in Initialize state. • The program is using protected resources that must be backed out. <p>System Action: The system returns this return code to the caller of the service.</p> <p>Application Programmer Response: Before it can use this conversation or any other protected conversations associated with the current context again, the local TP must issue a Backout call to restore all protected resources to their status as of the last synchronization point.</p>

Restrictions

TPs that call the Get_Attributes service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

ATBGETP - Get_TP_Properties (For MVS/ESA 4.2 through OS/390 V1R2)

Note: You cannot use the Error_Extract conversation service to diagnose errors in calls to the Get_TP_Properties service.

Equivalent to:

- LU 6.2 Get_TP_Properties
- (No CPI equivalent)

Returns information pertaining to the transaction program issuing the call.

This section describes the Get_TP_Properties (ATBGETP) callable service provided with MVS/ESA Version 4 Release 2. The Get_TP_Properties service was enhanced for OS/390 Release 8, and renamed ATBGTP4 (see “Get_TP_Properties” on page 175). The ATBGETP call remains valid, but does not contain the enhancements included in ATBGTP4.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBGETP(  
    Own_TP_name_length,  
    Own_TP_name,  
    Own_fully_qualified_LU_name,  
    User_id,  
    Profile,  
    LUW_id,  
    Return_code  
);
```

Figure 70. ATBGETP - LU 6.2 Get_TP_Properties

Parameters

Own_TP_name_length

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Own_TP_name_length specifies the length of data contained in the Own_TP_name parameter. If the Own_TP_name parameter does not contain a TP_name on return from this service, Own_TP_name_length is set to zero.

Own_TP_name

Returned parameter

- Type: Character string
- Char Set: 00640 or Type A
- Length: 64 bytes

Own_TP_name specifies the name of the local program as specified in the FMH-5 allocation request. This parameter will only contain a return value if the local program was started as the result of an attach request from a partner program. If this is not the case, there is no TP name to be returned, and Own_TP_Name_Length is set to zero.

Own_fully_qualified_LU_name

Returned parameter

- Type: Character string
- Char Set: Type A
- Length: 17 bytes

Own_fully_qualified_LU_name specifies the network-qualified name of the local logical unit.

User_id

Returned parameter

- Type: Character string
- Char Set: No restriction
- Length: 10 bytes

User_id specifies the user ID that is associated with the caller's address space. If the address space contains a scheduled transaction program, the User_id parameter contains the user ID that accompanied the inbound transaction program request.

Profile

Returned parameter

- Type: Character string

- Char Set: No restriction
- Length: 10 bytes

Profile specifies the RACF group name associated with the caller's address space. If the address space contains a scheduled transaction program, the profile parameter contains the RACF group name that accompanied the inbound transaction program request. If the inbound request did not include a profile, the profile parameter contains the default profile for the transaction program that issued the request.

LUW_id

Returned parameter

- Type: Structure
- Char Set: N/A
- Length: 26 bytes

LUW_id specifies the logical unit of work (LUW) identifier. The LUW identifier is used by some logical units for accounting purposes. If no LUW identifier is present, this field will contain binary zeroes.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. In cases where an error code is returned, the program should not examine any other returned variable associated with the call as nothing is placed in the variables.

See “Return Codes” on [page 473](#) for descriptions of return codes that can be returned to a caller of Get_TP_Properties.

Return Codes

Valid return code values for the Return_code parameter are:

Table 51. Return Codes for the Get_TP_Properties Service	
Return Code	Value, Meaning, and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_specific_error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write symptom records, which describe the error, to the logrec data set.</p> <p>Application Programmer Response: See “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>

Table 51. Return Codes for the Get_TP_Properties Service (continued)

Return Code	Value, Meaning, and Action
25	<p>Value: Program_state_check</p> <p>Meaning: The program called a service under conditions in which the call is not valid; for example:</p> <ul style="list-style-type: none"> • APPC/MVS might not recognize the program as a local TP. • APPC/MVS might have encountered temporary environmental conditions that prevent it from obtaining the requested information. <p>The program should not examine any other returned variables associated with the call because nothing is placed in those variables.</p> <p>System Action: The state of the conversation remains unchanged.</p> <p>Application Programmer Response: Design the program to re-issue the call; this error condition might be temporary.</p>

Restrictions

Transaction programs that call the Get_TP_Properties service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

ATBST05 - Set_TimeOut_Value (For OS/390 Release 8 through z/OS V1R6)

This section describes the Set_TimeOut_Value advanced callable service provided with provided with OS/390 Release 8. The Set_TimeOut_Value service was enhanced for z/OS V1R7, and renamed ATBST06 (see “Set_TimeOut_Value” on page 254). The ATBST05 call remains valid in z/OS V1R6, but does not contain the enhancements included in ATBST06.

Note: The ATBST06 call is the recommended programming interface for this service.

ATBST05 sets the time limit in minutes that each subsequent APPC/MVS conversation call will wait for VTAM APPCCMD requests to complete. For more information, see “Setting a Timeout Value for Potential Network Delays” on page 55.

The Set_Timeout_Value service can also be invoked to alter the previously set timeout_value.

For outbound transaction programs, the Set_TimeOut_Value service can be invoked at any time after the conversation is successfully established by the Allocate or CMINIT service.

For inbound transaction programs, the Set_TimeOut_Value service can be invoked at any time after successful completion of the Get_Conversation, Receive_Allocate, or CMACCP service.

Requirements

Authorization:	Supervisor state or problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts

Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Format

```
CALL ATBST05(
    Conversation_id,
    Timeout_Value,
    Return_code
);
```

Figure 71. ATBST05 - Set_TimeOut_Value

Parameters

Conversation_ID

Supplied parameter

- Type: Character string
- Char Set: N/A
- Length: 8 bytes

Conversation_ID specifies the conversation ID of the conversation for which you want to time VTAM APPCCMD requests issued during APPC/MVS conversation callable services. Specify the conversation_id that was returned from the Allocate, CMINIT, CMAACP, Get_Conversation, or Receive_Allocate call.

Timeout_value

Supplied parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits
- Value range: 0-1440 (decimal)

Specifies the time in minutes that all subsequent APPC/MVS conversation callable services will wait for VTAM APPCCMD requests to complete.

The maximum supported Timeout_Value is 1440 minutes (24 hours). When a Timeout_Value of zero is specified, VTAM APPCCMD requests issued by subsequent APPC/MVS conversation callable services will not be timed.

Any error in the specification of this parameter will result in a Program_Parameter_Check return code.

Return_code

Returned parameter

- Type: Integer
- Char Set: N/A
- Length: 32 bits

Return_code specifies the return code that is returned to the local program. Possible values of Return_code are:

Return Code	Value, Meaning and Action
0	<p>Value: OK</p> <p>Meaning: The call completed successfully.</p> <p>System Action: The system continues processing.</p> <p>Application Programmer Response: None required.</p>
20	<p>Value: Product_Specific_Error</p> <p>Meaning: The system found a product-specific error.</p> <p>System Action: The system might write the symptom records which describe the error to the logrec data set.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format. If necessary, see “Diagnosing Product-Specific Errors” on page 114 for more information about product-specific errors.</p>
24	<p>Value: Program_Parameter_Check</p> <p>Meaning: The system detected a program parameter check.</p> <p>System Action: The system returns this return code to caller of the APPC service in error.</p> <p>Application Programmer Response: See Chapter 6, “Diagnosing Problems with APPC/MVS TPs,” on page 77 for methods to use to diagnose the return code. See “Error_Extract” on page 158 for the Error_Extract calling format.</p>

Restrictions

Transaction programs that call the Set_Timeout_Value service while in task mode should not have any enabled unlocked task (EUT) functional recovery routines (FRRs) established. For more information about EUT FRRs, see the section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Appendix F. Accessibility

Accessible publications for this product are offered through [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the [Contact the z/OS team web page \(www.ibm.com/systems/campaignmail/z/zos/contact_z\)](http://www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element

The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element

The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE (KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**

The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

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 and IBM's Online Privacy Statement at 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.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, 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 is intended to help the customer to design and write APPC/MVS transaction programs. This documents General-use Programming Interface and Associated Guidance Information provided by z/OS.

General-use programming interfaces allow the customer to write programs that obtain the services of z/OS.

Trademarks

IBM, the IBM logo, and 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).

Index

Numerics

- 00640 character set
 - contents [387](#)
- 01134 character set
 - contents [387](#)

A

- ABEND processing [36](#)
- Accept_Test service
 - reference [262](#)
 - using [68](#)
- accessibility
 - contact IBM [477](#)
 - features [477](#)
- administrative system file
 - side information
 - overview [11](#)
 - TP profile
 - overview [11](#)
- advanced TP service [259](#)
- allocate queue service
 - description [11](#)
- allocate request
 - inbound
 - definition [6](#)
 - outbound
 - definition [6](#)
- Allocate service
 - ATBALC2 reference [433](#)
 - ATBALC5 [448](#)
 - reference [125](#), [425](#), [448](#)
 - using [47](#)
- API trace facility
 - avoiding loss of trace data
 - through suspension of tracing activity [85](#)
 - through wrapping [85](#)
 - compared with other diagnostic tools [78](#)
 - interpreting trace data [103](#)
 - overview [82](#)
 - security requirement [84](#)
 - setting up trace data sets [83](#)
 - starting tracing activity [86](#)
 - supported APPC/MVS and CPI-C calls [81](#)
 - trace data set characteristics [84](#)
 - using the ATBTRACE REXX exec
 - messages [93](#)
 - methods of invoking [94](#)
 - programming requirements [93](#)
 - restrictions [93](#)
 - return codes [93](#)
 - to list tracing activity [102](#)
 - to start tracing [99](#)
 - to stop tracing [101](#)
 - when to use [77](#), [78](#)
- APPC (Advanced Program-to-Program Communication)

- APPC (Advanced Program-to-Program Communication) (*continued*)
 - overview [3](#)
- APPC/MVS
 - callable service
 - advanced TP [259](#)
 - for CPI Communications [37](#)
 - callable services
 - combining CPI and MVS TP calls [32](#)
 - for TP conversations [125](#)
 - overview [9](#)
 - definition [9](#)
 - relation to APPC/VTAM [9](#)
 - server [13](#)
 - transaction scheduler [13](#)
- APPC/MVS allocate queue services
 - description [11](#)
- APPC/MVS server [13](#)
- APPCCMD macro
 - timeout considerations [55](#)
- applications for APPC/MVS [22](#)
- ASCII data
 - converting to EBCDIC [35](#)
- assembler programming language
 - call syntax [120](#)
- assistive technologies [477](#)
- asynchronous processing
 - overview [33](#)
 - using Notify_Type on MVS TP service [52](#)
- Asynchronous_Manager service
 - reference [259](#)
 - using [53](#), [64](#)
- ATBALC5
 - Allocate service [448](#)
- ATBTRACE REXX exec
 - issuing a LIST request
 - parameter descriptions [102](#)
 - syntax [102](#)
 - issuing a START request
 - parameter descriptions [99](#)
 - syntax [99](#)
 - issuing a STOP request
 - parameter descriptions [101](#)
 - syntax [101](#)
 - messages [93](#)
 - methods of invoking
 - from a high-level language program [94](#)
 - in MVS batch mode [94](#)
 - in TSO/E [94](#)
 - through TP profile JCL [94](#)
 - programming requirements [93](#)
 - restrictions [93](#)
 - return codes [93](#)

B

- benefits of APPC/MVS [21](#)

C

- C programming language
 - call syntax [120](#)
- call syntax
 - for APPC/MVS service [120](#)
 - for CPI Communications [40](#)
- callable services
 - advanced TP [259](#)
 - combining CPI Communications and MVS TP calls [32](#)
 - for CPI Communications [37](#)
 - for TP conversation [125](#)
 - overview [9](#)
- character set
 - used in APPC/MVS [387](#)
- Cleanup_TP--Unauthorized service
 - MVS/ESA SP 4.2.0 version reference [463](#)
 - MVS/ESA SP 4.2.2 version reference [263](#)
 - using [68](#)
- CMACCP (Accept_Conversation) [37](#)
- CMALLC (Allocate) [37](#)
- CMCFM (Confirm) [37](#)
- CMCFMD (Confirmed) [37](#)
- CMDEAL (Deallocate) [37](#)
- CMECT (Extract_Conversation_Type) [37](#)
- CMEMN (Extract_Mode_Name) [37](#)
- CMEPLN (Extract_Partner_LU_Name) [37](#)
- CMESL (Extract_Sync_Level) [37](#)
- CMFLUS (Flush) [37](#)
- CMINIT (Initialize_Conversation) [37](#)
- CMPTR (Prepare_to_Receive) [37](#)
- CMRCV (Receive) [37](#)
- CMRTS (Request_to_Send) [37](#)
- CMSCCT (Set_Conversation_Type) [37](#)
- CMSDT (Set_Deallocate_Type) [37](#)
- CMSD (Set_Error_Direction) [37](#)
- CMSEND (Send_Data) [37](#)
- CMSERR (Send_Error) [37](#)
- CMSF (Set_Fill) [38](#)
- CMSLD (Set_Log_Data) [38](#)
- CMSMN (Set_Mode_Name) [38](#)
- CMSPLN (Set_Partner_LU_Name) [38](#)
- CMSPTR (Set_Pprepare_to_Receive_Type) [38](#)
- CMSRC (Set_Return_Control) [38](#)
- CMSRT (Set_Receive_Type) [38](#)
- CMSL (Set_Sync_Level) [38](#)
- CMSST (Set_Send_Type) [38](#)
- CMSTPN (Set_TP_Name) [38](#)
- COBOL programming language
 - call syntax [120](#)
- concurrent APPC requests from one TP [31](#)
- Confirm service
 - reference [140](#)
 - using [51](#)
- confirmation
 - granting [24](#)
 - requesting [24](#)
- Confirmed service
 - reference [147](#)
 - using [51](#)
- contact
 - z/OS [477](#)
- contention between LUs
 - definition [9](#)

- contention between LUs (*continued*)
 - loser [9](#)
 - winner
 - specifying [129](#), [429](#), [438](#), [452](#)
- conversation
 - basic and mapped
 - overview [34](#)
 - specifying on Allocate service [47](#), [172](#)
 - definition for APPC [6](#)
 - ending
 - overview [24](#)
 - when errors occur [36](#)
 - flow diagram
 - confirmed transaction [28](#)
 - one-way conversation [4](#), [26](#)
 - sending error notification [29](#)
 - two-way conversation [27](#)
 - inbound
 - definition [6](#)
 - overview [14](#)
 - outbound
 - definition [6](#)
 - overview [13](#)
 - protected [35](#)
 - security [34](#)
 - services
 - map to APPC/MVS services [25](#), [26](#)
 - overview [23](#)
 - services of APPC/MVS [125](#)
 - starting
 - overview [23](#)
 - state
 - definition [6](#)
 - for CPI Communications [41](#)
 - for MVS TP service [45](#)
 - overview [23](#)
- conversation correlator
 - description [48](#)
- conversation_ID
 - definition [6](#)
- Conversation_ID parameter
 - on Error_Extract service [159](#)
- CPI (Common Programming Interface) Communications
 - call syntax [40](#)
 - calls supported by APPC/MVS
 - combining with MVS TP calls [32](#)
 - conversation states [41](#)
 - overview [37](#)
 - programming scenario [4](#)
 - relation to APPC [4](#)
 - transaction program environment [39](#)

D

- data
 - converting between ASCII and EBCDIC [35](#)
 - from the API trace facility [83](#)
 - granting permission to send
 - overview [24](#)
 - receiving
 - overview [23](#)
 - requesting permission to send
 - overview [24](#)
 - sending

- data (*continued*)
 - sending (*continued*)
 - overview [23](#)
- data set
 - for use with the API trace facility [83](#)
- data space
 - sending data [32](#)
- Deallocate service
 - reference [150](#)
 - using [52](#)
- diagnosis
 - interpreting API trace data [103](#)
 - problems with a transaction program [77](#)
 - starting API tracing activity [86](#)
 - tools for application programmers [77](#)

E

- EBCDIC data
 - converting to ASCII [35](#)
- ECB (event control block)
 - using with asynchronous APPC service [52](#)
- error information
 - returning [24](#)
- error log information
 - from Error_Extract service [109](#)
- error notification
 - sending
 - overview [24](#)
- Error_Extract service
 - calling [109](#)
 - calling for unestablished conversation [110](#)
 - compared with other diagnostic tools [78](#)
 - error log information [109](#)
 - example, synchronous call [110](#), [112](#)
 - for asynchronous calls [111](#)
 - overview [108](#)
 - product set ID [109](#)
 - reason codes [313](#)
 - reference [158](#)
 - supported APPC/MVS and CPI-C calls [81](#)
 - using [51](#)
 - when to use [77](#), [78](#)
- errors
 - diagnosing [77](#)
 - how to handle
 - design considerations [36](#)
- example
 - APPC/MVS transaction program [65](#)
- Extract_Information service
 - reference [266](#)
 - using [58](#)

F

- feedback [xxi](#)
- flow diagram of APPC conversation
 - confirmed transaction [28](#)
 - one-way conversation [4](#), [26](#)
 - sending error notification [29](#)
 - two-way conversation [27](#)
- Flush service
 - reference [163](#)

- Flush service (*continued*)
 - using [50](#)
- FORTAN programming language
 - call syntax [120](#)

G

- generic ID
 - using with multi-trans TP [61](#)
- generic user ID
 - using with multi-trans TP [63](#)
- Get_Attributes service
 - reference [165](#), [465](#)
 - using [48](#)
- Get_Conversation service
 - reference [171](#)
 - using [48](#)
- Get_TP_Properties service
 - reference [175](#), [471](#)
 - using [49](#)
- Get_Transaction service
 - reference [272](#)
 - using [60](#)
- Get_Type service
 - reference [179](#)
 - using [48](#)

H

- half-duplex communication [22](#)
- high level language
 - C programming language [120](#)
 - FORTAN programming language [120](#)
 - PL/I programming language [120](#)
 - REXX programming language [120](#)

I

- inbound
 - allocate request [6](#)
 - conversation [6](#)

J

- JES services available to TPs [32](#)

K

- keyboard
 - navigation [477](#)
 - PF keys [477](#)
 - shortcut keys [477](#)

L

- local LU
 - definition [7](#)
- local transaction program
 - definition [5](#)
- logical unit type 6.2 (LU 6.2)
 - relation to APPC [3](#)
- logon mode

- logon mode (*continued*)
 - definition [9](#)
- LU (logical unit)
 - definition [7](#)
 - local
 - definition [7](#)
 - partner
 - definition [7](#)
- LU 6.2 (logical unit type 6.2)
 - APPC/MVS services [125](#)
 - LU 6.2 verb to APPC/MVS service relationship [419](#)
 - option set supported by APPC/MVS [420](#)
 - relation to APPC [3](#)

M

- macro syntax
 - how to read [xvii](#)
- multi-trans schedule type
 - alternative use [61](#)
 - example [62](#)
 - overview [33](#)
 - use for transaction programs [60](#)
- multiple conversations within a program [31](#)

N

- navigation
 - keyboard [477](#)
- network delays, setting a timeout for [55](#)

O

- option set
 - SNA LU 6.2, supported by APPC/MVS [420](#)
- outbound
 - allocate request [6](#)
 - conversation [6](#)

P

- partner
 - transaction program [5](#)
- partner LU (logical unit)
 - definition [7](#)
- performance
 - for CPI Communication services [41](#)
 - for MVS TP service [56](#)
- permission to send data
 - granting
 - overview [24](#)
 - requesting
 - overview [24](#)
 - with Request_to_Send service [224](#)
- PL/I programming language
 - call syntax [120](#)
- portability of transaction programs [32](#)
- Post_on_Receive service
 - reference [181](#)
 - using [53](#)
- Prepare_to_Receive service
 - reference [185](#)
 - using [51](#)

- procedures
 - for writing transaction programs [16](#)
- protected conversation
 - design considerations [35](#)

R

- Receive_and_Wait service
 - reference [208](#)
 - using [50](#)
- Receive_Immediate service
 - reference [196](#)
 - using [50](#)
- recovery routine [36](#)
- Register_Test service
 - reference [274](#)
 - using [68](#)
- Reject_Conversation service
 - reference [277](#)
 - using [65](#)
- Request_to_Send service
 - reference [224](#)
 - using [51](#)
- return code
 - for APPC/MVS services [391](#)
 - for CPI Communications services [391](#)
- Return_Transaction service
 - reference [281](#)
 - using [63](#)
- REXX programming language
 - call syntax
 - for APPC/MVS service [120](#)
 - for CPI Communications [40](#)
- RPG programming language
 - call syntax
 - for CPI Communications [41](#)

S

- scheduler
 - definition [13](#)
- security for APPC
 - conversation security
 - overview [34](#)
 - specifying with the Allocate service [130](#), [430](#), [438](#), [452](#)
- security_none
 - specifying with the Allocate service [130](#), [430](#), [438](#), [452](#)
- Security_none
 - definition [34](#)
- security_pgm
 - specifying with the Allocate service [130](#), [430](#), [439](#), [453](#)
- security_same
 - specifying with the Allocate service [130](#), [430](#), [439](#), [453](#)
- Security_same
 - definition [34](#)
 - specifying with CPI Communications [34](#)
- Send_Data service
 - reference [227](#)
 - using [50](#)
- Send_Error service
 - reference [236](#)
 - using [51](#)

- sending to IBM
 - reader comments [xxi](#)
- server [13](#)
- services
 - APPC conversation
 - APPC/MVS services [25](#), [26](#)
 - overview [23](#)
- session
 - definition [7](#)
 - parallel [8](#)
- Set_Conversation_Accounting_Information service
 - reference [282](#)
 - using [59](#)
- Set_Syncpt_Options service
 - reference [248](#)
- Set_TimeOut_Value service
 - reference [254](#)
- Set_TimeOut_Value--Unauthorized service
 - z/OS V1R6 version reference [474](#)
- shared conversations across program boundaries [31](#)
- shortcut keys [477](#)
- side information
 - overview [12](#)
- SNA (systems network architecture)
 - relation to APPC [3](#)
- SRB mode support for a transaction program [32](#)
- standard schedule type
 - overview [33](#)
 - use for transaction programs [59](#)
- state
 - conversation [23](#)
- state table
 - how to use [399](#)
- steps
 - for writing transaction programs [16](#)
- subordinate address space
 - definition [16](#)
- summary of changes
 - z/OS MVS Programming: Writing Transaction Programs for APPC/MVS [xxiii](#)
- surrogate user IDs
 - Security_pgm parameter on allocate [130](#), [430](#), [439](#), [453](#)
- systems network architecture (SNA)
 - relation to APPC [3](#)

T

- timeout
 - description [55](#)
 - Set_TimeOut_Value service [254](#)
- TP message log
 - compared with other diagnostic tools [78](#)
 - when to use [77](#), [78](#)
- TP profile
 - overview [11](#)
- TP_ID
 - definition [6](#)
- trademarks [484](#)
- transaction initiator
 - definition [16](#)
- transaction program
 - advanced service [259](#)
 - characters used in name [387](#)
 - comparison of diagnostic tools [78](#)

- transaction program (*continued*)
 - conversation service [125](#)
 - debugging [77](#)
 - definition [5](#)
 - design considerations [29](#)
 - environment
 - for APPC/MVS services [119](#)
 - for CPI Communications [39](#)
 - environment in APPC/MVS [30](#)
 - installing [67](#)
 - local, definition [5](#)
 - naming [38](#)
 - partner, definition [5](#)
 - return codes [391](#)
 - return codes for CPI Communications services [391](#)
 - steps for writing [16](#)
 - testing [67](#)
 - writing
 - with APPC/MVS LU 6.2 services [45](#)
 - with CPI Communications [37](#)
- transaction schedule type
 - overview [33](#)
 - using multi-trans
 - alternative [61](#)
 - example [62](#)
 - using standard [59](#)
- transaction scheduler
 - definition [13](#)
 - interface, overview [13](#)
- TSO/E
 - services available to APPC/MVS transaction programs [32](#)
- type A character set
 - contents [387](#)

U

- Unregister_Test service
 - reference [285](#)
 - using [68](#)
- user interface
 - ISPF [477](#)
 - TSO/E [477](#)

V

- verb
 - LU 6.2 verb to APPC/MVS service relationship [419](#)
 - SNA definition [6](#)
- Version_Service service
 - reference [287](#)
- VTAM (Virtual Telecommunications Access Method)
 - relation to APPC [4](#)

Z

- z/OS MVS Programming: Writing Transaction Programs for APPC/MVS
 - summary of changes [xxiii](#)



Product Number: 5650-ZOS

SA23-1397-50

