webMethods EntireX

Broker ACI for Assembler

Version 10.1

October 2017

WEBMETHODS

This document applies to webMethods EntireX Version 10.1 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1997-2017 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: EXX-ACI-101-20191129ASSEMBLER

Table of Contents

1 About this Documentation ... 1
- Document Conventions ... 2
- Online Information and Support ... 2
- Data Protection ... 3

2 EntireX Broker ACI for Assembler ... 5
- Call Format ... 6
- Broker ACI Control Block Layout ... 7
- Broker ACI Control Block DSECT ... 11
- Using the Broker ACI with SSL/TLS ... 17
- ACI Examples and DSECTs ... 21

3 Writing Client and Server Applications ... 23
- Basic Concepts of Client and Server ... 24
- API-TYPE and API-VERSION ... 26
- LOGON and LOGOFF ... 26
- USER-ID and TOKEN ... 27
- Control Block Fields and Verbs ... 29
- Implementation of Client and Server Components ... 32
- Blocked and Non-blocked Broker Calls ... 33
- Conversational and Non-conversational Mode ... 36
- Managing Conversation Contexts ... 38
- Delayed SEND Function ... 41
- Timeout Parameters ... 42
- Data Compression ... 44
- Error Handling ... 45
- Using Send and Receive Buffers ... 47
- Tracing ... 49
- Transport Methods ... 50
- Variable-length Error Text ... 53
- Programmatically Turning on Command Logging ... 53

4 Writing Applications: Units of Work ... 55
- What is a Unit of Work? ... 56
- Control Block Fields and Verbs ... 57
- Client/Server Programming for Units of Work ... 60
- Client/Server Programming for a Persistent Unit of Work ... 62
- Client/Server Restart after System Failure ... 64

5 Writing Applications: Attach Server ... 65
- Implementing an Attach Server ... 66
- Implementing Servers started by an Attach Server ... 69

6 Writing Applications: Command and Information Services ... 71
- Accessing the Services ... 72
- Security with Command and Information Services ... 77
- Examples of Command Service ... 79

7 Writing Applications using EntireX Security ... 81

General Programming Considerations 82
Authentication 84
Authorization 87
8 Broker ACI Fields 89
Field Formats 90
Field Descriptions 90
9 Broker ACI Functions 105
Overview Table 107
Function Descriptions 108
Option Descriptions 115
ACI Field/Function Reference Table 117
Unique Message ID 119
10 Broker UOW Status Transition 123
Initial UOW Status: NULL | Received 124
Initial UOW Status: Accepted | Delivered | Postponed 125
Initial UOW Status: Processed | Timedout 126
Initial UOW Status: Cancelled | Discarded | Backedout 127
Legend for UOW Status Transition Table 128
Table of Column Abbreviations 128
11 Broker CIS Data Structures 129
Command Request Structure 131
Command Request Parameter Combinations 134
Common Header Structure for Response Data 138
Information Request Structure 140
Information Reply Structures 149

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at **http://documentation.softwareag.com**. The site requires credentials for Software AG's Product Support site Empower. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at **https://empower.softwareag.com/**.

You can find product information on the Software AG Empower Product Support website at **https://empower.softwareag.com**.

To submit feature/enhancement requests, get information about product availability, and download products, go to **Products**.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the **Knowledge Center**.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at **https://empower.softwareag.com/public_directory.asp** and give us a call.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at **http://techcommunity.softwareag.com**. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 EntireX Broker ACI for Assembler

- Call Format 6
- Broker ACI Control Block Layout 7
- Broker ACI Control Block DSECT 11
- Using the Broker ACI with SSL/TLS 17
- ACI Examples and DSECTs 21

Advanced communication interface. ACI-based programming is the base technology of EntireX. It uses a traditional Application Programming Interface (API) approach for conducting client/server and peer-to-peer dialog between distributed processes.

This chapter describes the EntireX Broker ACI from the perspective of the programming language Assembler.

Call Format

Calls to EntireX Broker use the following arguments:

1. The ACI control block is the first argument.
2. The send buffer is the second argument.
3. The receive buffer is the third argument.
4. The error text buffer is the last argument. It can provide a short text of the error code, if desired. Sufficient buffer length must be supplied to allow the standard 40-byte long message to be returned by EntireX Broker. For ACI version 9 and above, the error text buffer can be greater than 40 bytes as specified in the ACI field `ERRTEXT-LENGTH`.

The send buffer and the receive buffer addresses must always be provided. If they are not required by the selected EntireX Broker function, you can provide an address to a dummy CL1 field.

The API is called with a statement such as the following:

- Under z/OS and with all broker stubs, use standard IBM z/OS calling conventions. For example:

  ```
  BALR R14,R15
  ```

  R1 must point to the parameter list to be passed to broker stub. R13 must point to a 72-byte standard save area.

- Additionally, under z/OS, you can invoke broker stub CICS, using the following `EXEC CICS LINK` command. The length of the COMMAREA is always 24.

  ```
  EXEC CICS LINK PROGRAM('CICSETB') COMMAREA(MYCOMM) LENGTH(24)
  ```

  See CICSETB.

  The COMMAREA must specify an area in working storage with the following information:

  - 8-byte character field "`ETBCOMM*`"
  - one full word containing the address of the Broker control block
  - one full word containing the address of send buffer
  - one full word containing the address of receive buffer

- one full word containing the address of error text buffer

If using z/OS: in REGISTER15, the broker stub returns the last four bytes of the `ERROR-CODE` field in the Broker ACI control block, that is, the error number.

If CL8'00000000' (zeros) are returned in the `ERROR-CODE` field, the operation has been performed successfully. However, function results other than CL8'00000000' (zeros) do not necessarily indicate an error. See *Error Handling*.

Broker ACI Control Block Layout

The following table shows the broker fields in order of the physical layout of the broker ACI control block and provides a brief description of each field. The fields are described in more detail under *Broker ACI Fields*. See the actual layout for Assembler in ***Broker ACI Control Block DSECT*** below.

Note: Header files and examples are provided as models if you want to write your own ACI applications (see ***ACI Examples and DSECTs*** for location). The list below does not include unused fields that are for internal purposes only. Check the included header files for the full layout.

See *Broker ACI Fields* for more information.

Broker ACI Field	Assembler Definition	Description / Related Information		API Vers.	Notes
`API-TYPE`	`S$TYPE DS X`	API type.	See *`API-TYPE` and `API-VERSION`*	1	
`API-VERSION`	`S$VERS DS X`	API version.		1	
`FUNCTION`	`S$FCT DS X`	See *Broker ACI Functions* to perform the Broker function.		1	
`OPTION`	`S$OPT DS X`	See `OPTION`.		1	
	`S$RESERV DS CL16`	Reserved for future use.		1	1
`SEND-LENGTH`	`S$LSEND DS F`	Send length.	See *Using Send and Receive Buffers*	1	
`RECEIVE-LENGTH`	`S$LREC DS F`	Receive length.		1	
`RETURN-LENGTH`	`S$LRET DS F`	Return length.		1	
`ERRTEXT-LENGTH`	`S$LETXT DS F`	Error text length.		1	
`BROKER-ID`	`S$BID DS CL32`	Broker ID. See *Using the Broker ID in Applications*.		1	

Broker ACI Field	Assembler Definition	Description / Related Information	API Vers.	Notes
SERVER-CLASS SERVER-NAME SERVICE	S$CLASS DS CL32 S$SERVER DS CL32 S$SERVIC DS CL32	Service. See *Control Block Fields and Verbs*.	1	3, 5
USER-ID	S$USERID DS CL32	User ID. See *USER-ID and TOKEN*.	1	
PASSWORD	S$PASSWD DS CL32	Password. See *Authentication*.	1	4,5
TOKEN	S$TOKEN DS CL32	Reconnection token. See *USER-ID and TOKEN*.	1	3, 5
SECURITY-TOKEN	S$STOKEN DS CL32	Security token. See *Writing Applications using EntireX Security*.	1	4, 5
CONV-ID	S$CONVID DS CL16	Conversation ID. See *Conversational and Non-conversational Mode*.	1	3, 5
WAIT	S$WAIT DS CL8	Wait value. See *Blocked and Non-blocked Broker Calls*.	1	3, 5
ERROR-CODE	S$ERROR DS CL8	Error code. See *Error Handling* and *Error Messages and Codes*.	1	
ENVIRONMENT	S$ENV DS CL32	Pass additional information to *Translation User Exit*. For more information see ACI field ENVIRONMENT.	1	3, 5
ADCOUNT	S$ADCNT DS F	Attempted delivery count. See *Writing Applications: Units of Work*.	2	
USER-DATA	S$UDATA DS CL16	Conversation User Data. See *Managing Conversation Contexts*.	2	3, 5
Not used.	S$MSGID DS CL32	Message ID.	2	4, 5
Not used.	S$MSGTYP DS CL16	Message type.	2	3, 5
	S$PTIME DS CL8	Reserved for future use.	2	1, 3, 5
NEWPASSWORD	S$NPSWD DS CL32	New password. See *Authentication*.	2	4, 5
Not used.	S$ADERR DS CL8	Adapter error.	2	
CLIENT-UID	S$CLUID DS CL32	Client User ID. See *Writing Applications using EntireX Security*	2	
CONV-STAT	S$CNVST DS X	Conversation status. See *Conversational and Non-conversational Mode*.	2	

Broker ACI Field	Assembler Definition	Description / Related Information		API Vers.	Notes
STORE	S$STORE DS X	Persistence or non-persistence of a UOW. See *Writing Applications: Units of Work*.		2	
	S$STATUS DS X	Reserved for future use.		2	1
UOWSTATUS	S$UOWSTA DS X	UOW Status.	See *Writing Applications: Units of Work*.	3	3, 5
UWTIME	S$UOWTIM DS CL8	UOW lifetime.		3	3, 5
UOWID	S$UOWID DS CL16	UOW unique identifier.		3	3, 5
USTATUS	S$USRSTA DS CL32	User status		3	
UOW-STATUS-PERSIST	S$UOWSTP DS X	Multiplier for persistent status lifetime. See *Writing Applications: Units of Work*.		3	2
	S$RESVD2	Reserved for future use. Length must be 3 bytes.		3	
LOCALE-STRING	S$LOCS DS CL40	Locale string. To be used to override or provide a codepage name to tell the broker the encoding of the data. For more information see ACI field LOCALE-STRING.		4	
DATA-ARCH	S$DARCH DS XL1	Data architecture.		4	2
FORCE-LOGON	S$FLOG DS CL1	Override Broker AUTOLOGON. See *Authentication*.	See *Writing Applications using EntireX Security*.	6	
	S$ENCL DS XL1	Deprecated. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.		6	2
KERNELSECURITY	S$KERSEC DS CL1	Kernel security.		7	
COMMITTIME	S$COMTIM DS CL(LCOMTIM)	Commit time. See *Writing Applications: Units of Work*.		7	
COMPRESSLEVEL	S$COMP DS CL1	Compression level. See *Data Compression*.		7	

Broker ACI Field	Assembler Definition	Description / Related Information	API Vers.	Notes
	S$RESVD3	Reserved for future use. Length must be 114 bytes.	7	
	S$RESVD4	Reserved for future use. Length must be 6 bytes.	8	
UWSTAT-LIFETIME	S$UWSTAL DS CL8	Add value for persistent status lifetime. See UWSTAT-LIFETIME.	8	
	S$RES731 DS F	Reserved for future use.	9	
	S$RES732 DS F	Reserved for future use.	9	
	S$RES733 DS F	Reserved for future use.	9	
CLIENT-ID	S$CLID DS F	Returns to a server application the unique instance number of a client application. It is returned on receipt of a message (RECEIVE or SEND with WAIT).	9	
LOG-COMMAND	S$LOGCMD DS CL1	Log the current command. See also *Programmatically Turning on Command Logging*.	9	
CREDENTIALS-TYPE	S$CRDTYP DS CL1	Indicates the credentials type to be used to authenticate a user. The default is to use user ID and password.	9	
VARLIST-OFFSET	S$OVARLS DS F	Internal Software AG field.	10	
LONG-BROKER-ID-LENGTH	S$LLBID DS F	See LONG-BROKER-ID-LENGTH.	10	
MESSAGE-ID	S$NMSGID DS CL64	See *Unique Message ID* under *Broker ACI Functions* in the EntireX Broker ACI Programming documentation.	11	
CORRELATION-ID	S$CORID DS CL64	CORRELATION-ID.	11	
USE-SPECIFIED-MESSAGE-ID	S$USEMID DS X	Use supplied MESSAGE-ID for SEND.	11	
USE-SPECIFIED-CORRELATION-ID	S$USECID DS X	Send supplied CORRELATION-ID to Broker.	11	
	S$RES11 DS F	Reserved for future use.	11	

Notes:

1. Reserved for future use.
2. You must set this field to a low value (X'00') if you do not intend to use it.
3. The field is transmitted up to the first blank or low value (X'00'). It is not transmitted if the first character is a blank or a low value (X'00').
4. All trailing low values (X'00') are truncated. The field is not transmitted if the entire field is a low value (X'00').

5. If fields are not needed for a specific command function, suppress their transmission by initializing them to blanks or low value (X'00').

Broker ACI Control Block DSECT

EntireX provides a DSECT with the ACI control block definition. See under *ACI Examples and DSECTs* where it is provided on your platform.

```
************************************************************************
* ENTIRE BROKER API Control Block Definition for use with
* IBM/370 or SIEMENS Assembler Language.
************************************************************************
* Product        : EntireX Broker
* Copyright      : Copyright (c) 1997 - 2017 Software AG, Darmstadt,
*                  Germany and/or Software AG USA, Inc., Reston, VA,
*                  United States of America, and/or their licensors.
* Version        : 10.1
* File Version   : $Revision: 1.90 $
* File           : ASMDEF
* Description    : Assembler control block definitions.
************************************************************************
*
* --- ENTIRE BROKER API Type Constants --------------------------------
*
* Please note, these are bit settings
*
$TYPE1   EQU   1                       Normal call type
$TYPE2   EQU   2                       Not used
$TYPE4   EQU   4                       Not used
$TYPE8   EQU   8                       Call could be ReplyError
*
* --- ENTIRE BROKER API Version Constants -----------------------------
*     For a new API version add in a new equate and increase $VERMAX
*
$VERS1   EQU   1
$VERS2   EQU   2
$VERS3   EQU   3
$VERS4   EQU   4
$VERS5   EQU   5
$VERS6   EQU   6
$VERS7   EQU   7
$VERS8   EQU   8
$VERS9   EQU   9
$VERS10  EQU   10
$VERS11  EQU   11
$VERMIN  EQU   $VERS1
$VERAMAX EQU   $VERS11                 Maximum Asm Stub Version
```

```
$VERMAX  EQU   $VERS11                 Maximum Kernel Version
*
* --- ENTIRE BROKER API Function Constants ----------------------------
*
$FCTSEND EQU   1                       SEND
$FCTREC  EQU   2                       RECEIVE
$FCTUNDO EQU   4                       UNDO
$FCTEOC  EQU   5                       EOC
$FCTREG  EQU   6                       REGISTER
$FCTDREG EQU   7                       DEREGISTER
$FCTVERS EQU   8                       VERSION
$FCTLOGN EQU   9                       LOGON
$FCTLOGF EQU   10                      LOGOFF
$FCTSET  EQU   11                      SET
$FCTGET  EQU   12                      GET
$FCTSYNC EQU   13                      SYNCPOINT
$FCTKVER EQU   14                      KERNELVERS
$FCTRPLY EQU   22                      REPLY ERROR
$FCTGMID EQU   26                      GET MESSAGE ID
*
* --- ENTIRE BROKER API Option Constants ------------------------------
*
$OPTOFF  EQU   0                       OFF
$OPTMSG  EQU   1                       MSG
$OPTHOLD EQU   2                       HOLD
$OPTIMME EQU   3                       IMMED
$OPTQUIE EQU   4                       QUIESCE
$OPTEOC  EQU   5                       EOC
$OPTCANC EQU   6                       CANCEL
$OPTLAST EQU   7                       LAST
$OPTNEXT EQU   8                       NEXT
$OPTPREV EQU   9                       PREVIEW
$OPTCOMM EQU   10                      COMMIT
$OPTBACK EQU   11                      BACKOUT
$OPTSYNC EQU   12                      SYNCPOINT
$OPTATT  EQU   13                      ATTACH
$OPTDEL  EQU   14                      DELETE
$OPTEOCC EQU   15                      EOC CANCEL
$OPTQRY  EQU   16                      QUERY
$OPTSETS EQU   17                      SET STATUS
$OPTANY  EQU   18                      ANY
$OPTTERM EQU   19                      TERMINATE
$OPTCSCV EQU   21                      CHECK SERVICE
*
* --- ENTIRE BROKER API Conversation Status Constants -----------------
*
$CNVNEW  EQU   1                       NEW
$CNVOLD  EQU   2                       OLD
$CNVNONE EQU   3                       NONE
*
* --- ENTIRE BROKER API Store Constants -------------------------------
*
```

```
$STROFF  EQU   1                        OFF
$STRBROK EQU   2                        BROKER
*
* --- ENTIRE BROKER API Status Constants ------------------------------
*
$STAOFF  EQU   1                        OFF
$STASTOR EQU   2                        STORED
$STADELA EQU   3                        DELIVERY ATTEMPTED
$STADELD EQU   4                        DELIVERED
$STAPROC EQU   5                        PROCESSED
$STADEAD EQU   6                        DEAD
*
* --- ENTIRE BROKER API UWSTATUS Options ------------------------------
*
$UWSNONE EQU   0                        RECEIVE NONE
$UWSRCVD EQU   1                        RECEIVED
$UWSACCP EQU   2                        ACCEPTED
$UWSDELV EQU   3                        DELIVERED
$UWSBOUT EQU   4                        BACKEDOUT
$UWSPROC EQU   5                        PROCESSED
$UWSCANC EQU   6                        CANCELLED
$UWSTOUT EQU   7                        TIMEOUT
$UWSDISC EQU   8                        DISCARDED
$UWSRCVF EQU   9                        RECV_FIRST
$UWSRCVM EQU   10                       RECV_MIDDLE
$UWSRCVL EQU   11                       RECV_LAST
$UWSRCVO EQU   12                       RECV_ONLY
$UWSPPON EQU   13                       POSTPONED
*
* --- ENTIRE BROKER API Architecture Constants ------------------------
*
$ACOHAI  EQU   1                        HIGH ASCII IBM
$ACOLAI  EQU   2                        LOW ASCII IBM
$ACOHEI  EQU   3                        HIGH EBCDIC IBM
$ACOLEI  EQU   4                        LOW EBCDIC IBM
$ACOHAV  EQU   5                        HIGH ASCII VAX
$ACOLAV  EQU   6                        LOW ASCII VAX
$ACOHEV  EQU   7                        HIGH EBCDIC VAX
$ACOLEV  EQU   8                        LOW EBCDIC VAX
$ACOHAE  EQU   9                        HIGH ASCII IEEE
$ACOLAE  EQU   10                       LOW ASCII IEEE
$ACOHEE  EQU   11                       HIGH EBCDIC IEEE
$ACOLEE  EQU   12                       LOW EBCDIC IEEE
*
* --- ENTIRE BROKER API Force Logon values ----------------------------
*
FLOGYES  EQU   C'Y'                     FORCE LOGON YES
FLOGNO   EQU   C'N'                     FORCE LOGON NO
FLOGS    EQU   C'S'                     FORCE LOGON S
*
* --- ENTIRE BROKER API -----------------------------------------------
*
```

```
ENCNONE  EQU   0
ENCBROK  EQU   1                       DEPRECATED
ENCTRGT  EQU   2                       DEPRECATED
*
* --- ENTIRE BROKER API Kernel Security -------------------------------
*
KSECNO   EQU   C'N'                    INDICATES NO KERNEL SECURITY
KSECYES  EQU   C'Y'                    INDICATES KERNEL SECURITY ON
KSECUSER EQU   C'U'                    INDICATES KERNEL SECURITY USER
KSECLITE EQU   C'L'                    INDICATES KERNEL SECURITY LIGHT
*
* --- ENTIRE BROKER API Compression Level -----------------------------
*
$COMP0   EQU   C'0'                    COMPRESSION LEVEL = 0
$COMP1   EQU   C'1'                    COMPRESSION LEVEL = 1
$COMP2   EQU   C'2'                    COMPRESSION LEVEL = 2
$COMP3   EQU   C'3'                    COMPRESSION LEVEL = 3
$COMP4   EQU   C'4'                    COMPRESSION LEVEL = 4
$COMP5   EQU   C'5'                    COMPRESSION LEVEL = 5
$COMP6   EQU   C'6'                    COMPRESSION LEVEL = 6
$COMP7   EQU   C'7'                    COMPRESSION LEVEL = 7
$COMP8   EQU   C'8'                    COMPRESSION LEVEL = 8
$COMP9   EQU   C'9'                    COMPRESSION LEVEL = 9
$COMPY   EQU   C'Y'                    COMPRESSION LEVEL = Y
$COMPN   EQU   C'N'                    COMPRESSION LEVEL = N
*
* --- ENTIRE BROKER API Size Of Fields --------------------------------
*
LPUID    EQU   28                      L'ADABAS Extended UserID
LPLTFRM  EQU   8                       L'PLATFORM String
LTRNAME  EQU   8                       L'Translation Module Name
LTXTOK   EQU   64                      L'TxToken
LTXCLTOK EQU   64                      L'TxClientToken
LCOMTIM  EQU   17                      L'Commit Time
LMSGID   EQU   64                      L'MessageID
*
* --- ENTIRE BROKER API Control Block ---------------------------------
*
*     For a new version of the ETBCB, add an equate to the end to
*     indicate the length of the previous version, then add to the
*     end. The length field for a version is LETBCBx where "x"
*     is the version name.  The current version length is LETBCB.
*
ETBCB    DSECT ,
S$TYPE   DS    X                       API-TYPE
S$VERS   DS    X                       API-VERSION
S$FCT    DS    X                       FUNCTION
S$OPT    DS    X                       OPTION
S$RESERV DS    CL16                    RESERVED
S$LSEND  DS    F                       SEND-LENGTH
S$LREC   DS    F                       RECEIVE-LENGTH
S$LRET   DS    F                       RETURN-LENGTH
```

```
S$LETXT  DS    F                        ERRTEXT-LENGTH
S$BID    DS    CL32                     BROKER-ID
S$CLASS  DS    CL32                     SERVER-CLASS
S$SERVER DS    CL32                     SERVER-NAME
S$SERVIC DS    CL32                     SERVICE
S$USERID DS    CL32                     USER-ID
S$PASSWD DS    CL32                     PASSWORD
S$TOKEN  DS    CL32                     TOKEN
S$STOKEN DS    CL32                     SECURITY-TOKEN
S$CONVID DS    CL16                     CONV-ID
S$WAIT   DS    CL8                      WAIT
S$ERROR  DS   0CL8                      ERROR-CODE
S$ERRCL  DS    CL4                      ERROR-CLASS
S$ERRNR  DS    CL4                      ERROR-NUMBER
S$ENV    DS    CL32                     ENVIRONMENT
LETBCB1  EQU   *-ETBCB             --- VERSION 1 LENGTH
S$ADCNT  DS    F                        ATTEMPTED DELIVERY COUNT
S$UDATA  DS    CL16                     USER DATA FIELD
S$MSGID  DS    CL32                     NOT USED BY BROKER
S$MSGTYP DS    CL16                     NOT USED BY BROKER
S$PTIME  DS    CL8                      NOT USED BY BROKER
S$NPSWD  DS    CL32                     NEW PASSWORD OF CALLER
S$ADERR  DS    CL8                      ADAPTER ERROR
S$CLUID  DS    CL32                     CLIENT USER-ID
S$CNVST  DS    X                        CONVERSATION STATUS
S$STORE  DS    X                        STORE MESSAGE
S$STATUS DS    X                        NOT USED BY BROKER
S$UOWSTA DS    X                        STATUS OF UNIT-OF-WORK
LETBCB2  EQU   *-ETBCB             --- VERSION 2 LENGTH
S$UOWTIM DS    CL8                      LIFETIME OF UOW
S$UOWID  DS    CL16                     UNIT-OF-WORK ID
S$USRSTA DS    CL32                     USER STATUS
S$UOWSTP DS    X                        UOW STATUS PERSIST FLAG
S$RESVD2 DS    XL3                      RESERVED
         DS    OF                       ALIGNMENT
LETBCB3  EQU   *-ETBCB             --- VERSION 3 LENGTH
S$LOCS   DS    CL40                     LOCALE STRING
S$DARCH  DS    XL1                      FOR FUTURE USE
S$FLOG   DS    CL1                      FORCE LOGON
S$ENCL   DS    XL1                      DEPRECATED - USE AT-TLS
S$KERSEC DS    CL1                      KERNEL SECURITY
LETBCB4  EQU   *-ETBCB             --- VERSION 4 LENGTH
S$COMTIM DS    CL(LCOMTIM)              COMMIT_TIME
S$COMP   DS    CL1                      COMPRESSION LEVEL
S$RESVD3 DS    CL114
LETBCB6  EQU   *-ETBCB             --- VERSION 6 LENGTH
LETBCB7  EQU   *-ETBCB             --- VERSION 7 LENGTH
         ORG   S$RESVD3
S$RESVD4 DS    CL6
S$UWSTAL DS    CL8                      UOW STATUS LIFE TIME (V8)
S$RES911 DS    CL96                     RESERVED FOR FUTURE USE
S$RES912 DS    CL16                     RESERVED FOR FUTURE USE
```

```
LETBCB8  EQU   *-ETBCB             --- VERSION 8 LENGTH
S$RES991 DS    CL32                    RESERVED FOR FUTURE USE
S$RES731 DS    F                       RESERVED FOR FUTURE USE
S$RES732 DS    F                       RESERVED FOR FUTURE USE
S$RES733 DS    F                       RESERVED FOR FUTURE USE
S$CLID   DS    F                       CLIENT_ID
S$RES734 DS    CL32                    RESERVED FOR FUTURE USE
S$LOGCMD DS    CL1                     LOG THIS COMMAND
S$CRDTYP DS    CL1                     CREDENTIALS TYPE
S$RES735 DS    CL32                    RESERVED FOR FUTURE USE
         DS    CL2                     ALIGNMENT
LETBCB9  EQU   *-ETBCB             --- VERSION 9 LENGTH
S$OVARLS DS    F                       VARIABLE LIST OFFSET
S$LLBID  DS    F                       LENGTH LONG BROKER-ID
LETBCB10 EQU   *-ETBCB             --- VERSION 10 LENGTH
S$NMSGID DS    CL(LMSGID)              MSG ID OF SENT/RCVD MSG
S$CORID  DS    CL(LMSGID)              MSG ID OF SENT MSG
S$USEMID DS    X                       No new MSG ID for SEND
S$USECID DS    X                       Send COR ID to Broker
         DS    CL2                     ALIGNMENT
S$RES11  DS    F                       RESERVED FOR FUTURE USE
LETBCB11 EQU   *-ETBCB             --- VERSION 11 LENGTH
         ORG   ,
LETBCB   EQU   *-ETBCB                 ETBCB LENGTH
*
* --- ENTIRE BROKER Attach Manager Control Block ----------------------
*
ETBATMCB DSECT ,
A$VERS   DS    H                       VERSION OF STRUCTURE
         DS    H                       ALIGNMENT
A$NATTCH DS    F                       NO. FAILED SERVER LOOKUPS
A$NSERVR DS    F                       NO. REGISTERED SERVER
A$NCPEND DS    F                       NO. PENDING CONVERSATIONS
A$NCACT  DS    F                       NO. ACTIVE CONVERSATIONS
A$CLASS  DS    CL(L'S$CLASS)           CLASS NAME TO ATTACH
A$SERVER DS    CL(L'S$SERVER)          SERVER NAME TO ATTACH
A$SERVIC DS    CL(L'S$SERVIC)          SERVICE NAME TO ATTACH
         DS    OH                      ALIGNMENT
LETBATMC EQU   *-ETBATMCB              ETBATMCB LENGTH
*
* --- End of definitions
```

Using the Broker ACI with SSL/TLS

ACI applications can use Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium. The term “SSL” in this section refers to both SSL and TLS. ACI-based clients or servers are always SSL clients. The SSL server can be either the EntireX Broker or the Broker SSL Agent. For an introduction see *SSL/TLS and Certificates with EntireX* in the EntireX Security documentation. This section describes using the Broker ACI with SSL on the following platforms:

- z/OS
- z/VSE

z/OS

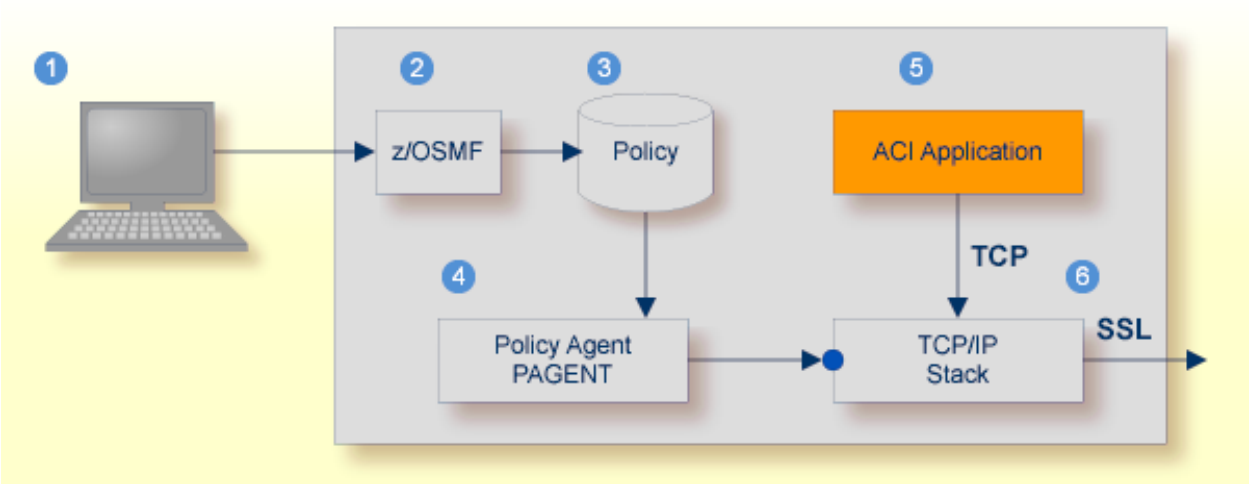
SSL delivered on a z/OS mainframe will typically use the Resource Access Control Facility (RACF) as the certificate authority (CA). Certificates managed by RACF can only be accessed through the RACF keyring container. A keyring is a collection of certificates that identify a networking trust relationship (also called a trust policy). In an SSL client/server network environment, entities identify themselves using digital certificates called through a keyring. Server applications on z/OS that wish to establish network connections to other entities can use keyrings and their certificate contents to determine the trustworthiness of the client or peer entity. Note that certificates can belong to more than one keyring, and you can assign different users to the same keyring. Because of the way RACF internally references certificates, they must be uniquely identifiable by owner and label, and also unique by serial number plus data set name (DSN).

For establishing an SSL connection on z/OS, IBM's Application Transparent Transport Layer Security (AT-TLS) can be used, where the establishment of the SSL connection is pushed down the stack into the TCP layer.

Using IBM's Application Transparent Transport Layer Security (AT-TLS)

With the Broker ACI for Assembler you can use IBM's Application Transparent Transport Layer Security, where the establishment of the SSL connection is pushed down the stack into the TCP layer.

Configure the AT-TLS rules for the policy agent (PAGENT) 4 using an appropriate client 1 and the z/OS Management Facility (z/OSMF) 2. Together with SSL parameters (to provide certificates stored in z/OS as RACF keyrings) define AT-TLS rules, for example by using the application 5 job name and remote TCP port number. If the rules match, the TCP connection is turned into an SSL connection 6. Refer to your IBM documentation for more information, for example the IBM Redbook *Communications Server for z/OS VxRy TCP/IP Implementation Volume 4: Security and Policy-Based Networking*.



1. Client to interact with z/OS Management Facility (z/OSMF).
2. AT-TLS rules are defined with z/OSMF policy management.
3. Policy Repository with AT-TLS rules stored as z/OS files.
4. Policy Agent, MVS task PAGENT, provides AT-TLS rules through a policy enforcement point (PEP) to TCP/IP stack.
5. Application using TCP connection.
6. If AT-TLS rules match, the TCP connection is turned into an SSL connection.

Notes:

1. The client 1 may vary per operating system, for example a Web browser for z/OS 2.1.
2. z/OSMF 2 includes other administration and management tasks in addition to policy management.
3. Policy Management 3 includes other rules, such as IP filtering, network address translation etc.

To set up SSL with AT-TLS

1 To operate with SSL, certificates need to be provided and maintained. Depending on the platform, Software AG provides default certificates, but we strongly recommend that you create your own. See *SSL/TLS Sample Certificates Delivered with EntireX* in the EntireX Security documentation.

2 Set up the ACI application (client or server) for a TCP/IP connection. On mainframe platforms, use *Transport-method-style Broker ID*. Example

```
ETB024:1699:TCP
```

3 Configure AT-TLS to turn the TCP/IP connection to an SSL connection, see above.

4 Make sure the SSL server to which the ACI application (client or server) connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in Integration Server (IS inbound). See:

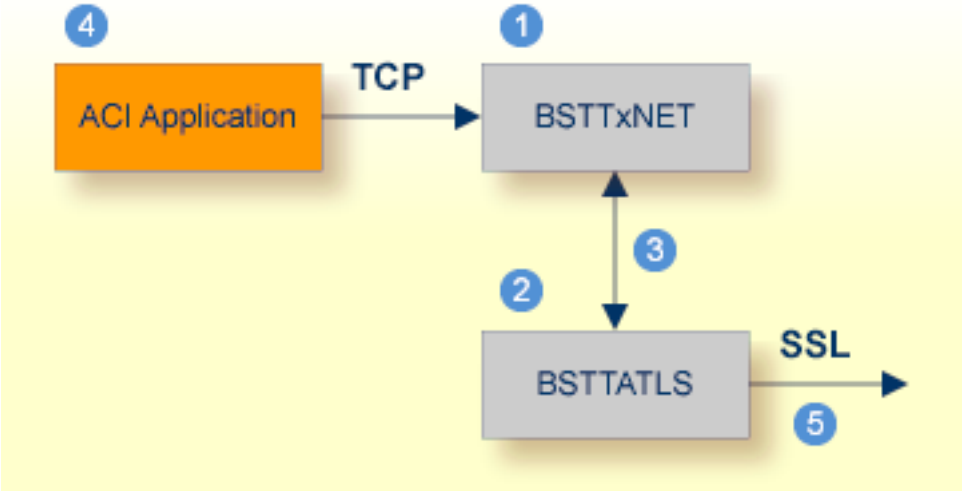
- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation
- *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

z/VSE

Establishing an SSL connection on z/VSE requires BSI's Automatic Transport Layer Security (ATLS). This facility is similar to z/OS Application Transparent - Transport Layer Security (AT-TLS). ATLS is supported by the BSI stack only.

Using BSI's Automatic Transport Layer Security (ATLS)

Together with SSL parameters (to provide certificates), define ATLS rules for socket interception in the ATLS daemon startup job `BSTTATLS` 2. If the rules match, the socket connection is turned into an SSL connection 5. Refer to your IBM documentation for further information. For an overview, refer to the IBM Redbook *Enhanced Networking on IBM z/VSE*; for a more detailed description, refer to *BSI SSL Installation, Programming and User's Guide*.



1 BSI TCP/IP Stack, either BSTTINET (IPv4) or BSTT6NET (IPv6).

2. ATLS rules are defined manually. See Sample ATLS Daemon Configuration below.
3. BSTTATLS is associated with a TCP/IP stack.
4. Application using TCP connection.
5. BSTTATLS intercepts outbound TCP connection and converts it to SSL connection. For inbound, SSL connections can also be intercepted and converted to TCP connections.

To set up SSL with AT-TLS

1. To operate with SSL, certificates need to be provided and maintained. Depending on the platform, Software AG provides default certificates, but we strongly recommend that you create your own. See *SSL/TLS Sample Certificates Delivered with EntireX* in the EntireX Security documentation.
2. Set up the RPC component for a TCP/IP connection. On mainframe platforms, use *Transport-method-style Broker ID*. Example:

```
ETB024:1699:TCP
```

3. Configure ATLS to turn the TCP/IP connection to an SSL connection, see above.
4. Make sure the SSL server to which the RPC component connects is prepared for SSL connections as well. The SSL server can be EntireX Broker, Broker SSL Agent, or Direct RPC in webMethods Integration Server (IS inbound). See:
   - *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
   - *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation
   - *Support for SSL/TLS* in the EntireX Adapter documentation (for Direct RPC)

Sample ATLS Daemon Configuration

```
* Converting inbound EntireX Broker connection
* Converts listen port 1971 to SSL listen port 1972
OPTION SERVER
ATTLS 1971 AS 2071 SSL
*
* Converting outbound client connection
* Converts connect to 192.168.2.100:1972:TCP to 192.168.2.100:2072:SSL
OPTION CLIENT
ATTLS 1972 TO 192.168.2.100 AS 2072 SSL
```

Note: We recommend setting `SETPARM` value `SUBTASK` to a value greater than 0 in the ATLS daemon startup job (valid values 0-16, default=0). For example:

```
// SETPARM SUBTASK=8
```

See also *BSI SSL Installation, Programming and User's Guide*.

For additional information see also *Using the BSI TCP/IP Stack* under *Hints for Setting up Broker JCL in z/VSE* in the z/VSE Installation documentation and *SSL/TLS and Certificates with EntireX*.

ACI Examples and DSECTs

Depending on your platform for Assembler, you will find the files with the examples, DSECTs, etc., at the following locations:

Platform	DSECTs / Examples	Location	Note
z/OS	Broker ACI control block DSECT	See member `ASMDEF` in the mainframe source library EXX101.SRCE.	1
	Broker Command and Info Services control block DSECT	See member `ASMINF` in the mainframe source library EXX101.SRCE.	
BS2000	Broker ACI control block DSECT	See element `ASMDEF` in the LMS library EXX101.LIB.	
	Broker Command and Info Services control block DSECT	See element `ASMINF` in the LMS library EXX101.LIB.	

Notes:

1. See *Installing EntireX under z/OS*.

3 Writing Client and Server Applications

- Basic Concepts of Client and Server 24
- API-TYPE and API-VERSION 26
- LOGON and LOGOFF 26
- USER-ID and TOKEN 27
- Control Block Fields and Verbs 29
- Implementation of Client and Server Components 32
- Blocked and Non-blocked Broker Calls 33
- Conversational and Non-conversational Mode 36
- Managing Conversation Contexts 38
- Delayed SEND Function 41
- Timeout Parameters 42
- Data Compression 44
- Error Handling 45
- Using Send and Receive Buffers 47
- Tracing 49
- Transport Methods 50
- Variable-length Error Text 53
- Programmatically Turning on Command Logging 53

The client and server communication model is based on a logical connection between exactly two partners: a client and a server. It covers the communication requirements conversational and non-conversational, and synchronous and asynchronous. This chapter describes how to implement and program client and server applications with EntireX Broker.

See also *Writing Applications: Attach Server* and *Writing Applications: Units of Work*.

Basic Concepts of Client and Server

- Client-and-Server Application Components
- Conversationality
- Synchronicity

Client-and-Server Application Components

In the client-and-server communication model there are two partner application components: a requesting partner (the client) and the partner satisfying the request (the server). The client identifies the required service through the names of the `SERVER-CLASS`, `SERVER-NAME` and `SERVICE` with which the partner has registered.

EntireX Broker allows multiple server application components to register the same service in order to satisfy processing requirements. In conversational requests, the client and the server are bound to each other for the duration of the conversation. In addition, a server application component can satisfy more than one request type after registering several class, server and service names.

An application component is not restricted to a single role as either client or server; it can perform the role of both client and server. It can therefore make requests for processing while also satisfying requests from other partner application components.

Conversationality

The EntireX Broker allows both non-conversational and conversational communication in order to meet the different requirements of connections between distributed application components.

- **Non-conversational**
  In this communication type, each request comprises a single message from the client that requires at most one reply from a server. Since there is only one `SEND` / `RECEIVE` cycle per request, each request can be satisfied individually by any of a number of server replicas.
- **Conversational**
  In this communication type, the request contains a series of related messages, initiated by a client, which occur between client and server. Since there is a series of `SEND` / `RECEIVE` commands for each request, the same replica of a server must process all related messages within a conversation.

Using EntireX Broker, an application may have more than one conversation active at the same time with the same partner or with different partners. Conversational and non-conversational modes can also be used simultaneously. The required mode of communication is always controlled by the application component that initiates the communication, that is, the client side.

Synchronicity

EntireX Broker makes possible both synchronous and asynchronous communication. EntireX Broker enables application components to combine synchronous and asynchronous communication as needed by the application. The terms synchronous and asynchronous correspond to the terms “blocked” and “non-blocked”. See *Blocked and Non-blocked Broker Calls*.

- **Synchronous**
  The application component initiating the request waits for the processing to be completed by the partner application component before continuing. EntireX Broker provides the application with facilities to wait automatically for the partner application to complete processing and reply to the requesting application partner.
- **Asynchronous**
  The application component initiating the request does not wait for the processing to be completed and continues to execute without needing to receive a reply from the partner application. EntireX Broker provides the application with facilities to continue processing and obtain the partner's reply at a later time, if needed.

API-TYPE and API-VERSION

Both the API-TYPE and the API-VERSION fields must always be provided.

Value	Bit Pattern	Description
1	(x'01')	The standard value for API-TYPE is 1 (x'01') and usable with all Broker stubs in all environments. **Note:** If any of the following conditions exist, you must install the Adabas CICS link module with the definition PARMTYP=ALL, using the ADAGSET macro. 1. If you are using NET transport with CICSETB stub with send or receive buffers greater than 32 KB. 2. If you are using NET transport with CICSETB stub and your application does not have a TWA.

Certain Broker functionality requires a minimum API-VERSION. For the highest available version of Broker, see API-VERSION. The send buffer and the receive buffer are passed as parameters to the EntireX Broker. Both buffers can occupy the same location.

See *Broker ACI Control Block Layout* in the EntireX language-specific ACI documentation.

Both the API-TYPE and API-VERSION fields must be set correctly to ensure that Broker returns the correct value in ACI field ERROR-CODE. Otherwise, depending on your programming language and environment, a return code may not always be given.

See *Call Format* in the language-specific EntireX ACI documentation.

LOGON and LOGOFF

The LOGON and LOGOFF Broker functions are optional when using the client-and-server programming model in your application. However, we recommend that the application issues LOGON and LOGOFF function calls for the following reasons:

- LOGOFF will notify the Broker to clean up in-memory resources held for your program, making them available to other users of the Broker.
- Without LOGOFF, the user's in-memory resources will time out in accordance with the Broker attributes CLIENT-NONACT and SERVER-NONACT. Depending on the values set by the administrator, this may not occur for some time.

Example for programming language Natural:

```
/* Logon to Broker/LOGON
MOVE #FCT-LOGON TO #ETBCB.#FUNCTION
/*
CALL 'BROKER' #ETBAPI #SEND-BUFF #RECV-BUFF #ERR-TXT
```

Logoff example for programming language Natural:

```
/* Logoff to Broker/LOGOFF
MOVE #FCT-LOGOFF TO #ETBCB.#FUNCTION
CALL 'BROKER' #ETBAPI #SEND-BUFF #RECV-BUFF #ERR-TXT
```

USER-ID and TOKEN

- Identifying the Caller
- Restarting after System Failure
- Managing the Security Token

Identifying the Caller

`USER-ID` identifies the caller and is required for all functions except `VERSION`. The `USER-ID` is combined with an internal ID or with the `TOKEN` field, if supplied, in order to guarantee uniqueness, for example where more than one application component is executing under a single `USER-ID`.

Brokers identify callers as follows:

- When the ACI field `TOKEN` is supplied:

  The ACI field `USER-ID`, together with the `TOKEN`, is used to identify the user. Using `TOKEN` allows the application to reconnect with a different process or thread without losing the existing conversation. When a new call is issued under the same `USER-ID` from a different location but with the same `TOKEN`, the caller is reconnected to the previous context.

  Note: The ability to reconnect to the previous context is vital if restart capabilities of applications are required. The combination of `USER-ID` and `TOKEN` must be unique to the Broker. It is not possible to have the same `USER-ID` and `TOKEN` combination duplicated.

- When the ACI field `TOKEN` is not supplied:

  The `USER-ID` is combined with an internally generated ID. It is possible to use the same `USER-ID` in different threads or processes. All threads and processes are distinct Broker users.

Restarting after System Failure

The Broker provides a reconnection feature, using the `TOKEN` field in the ACI. If the application supplies a token along with `USER-ID`, the processing is automatically transferred when a request with the same user ID and token is received, either from the same process or from a different process or thread.

Specification of `USER` and `TOKEN` is necessary for reconnection with the correct user context after Broker has been stopped and restarted.

Managing the Security Token

If you are using EntireX Security, the application must maintain the content of the `SECURITY-TOKEN` field and not change this field on subsequent calls.

Control Block Fields and Verbs

- Basic Functionality of Broker API
- ACI Syntax
- Key ACI Field Names
- Key Verbs for FUNCTION Field

Basic Functionality of Broker API

This section describes the basic functionality of the Broker API. The following functions in the Broker API are fundamental to client-and-server processing. For full set of verbs relating to UOW processing, see *Control Block Fields and Verbs*.

- `DEREGISTER`
  The function `DEREGISTER` is used by a server to indicate its intention to terminate its role as a server for the named `SERVER-CLASS`, `SERVER-CLASS` and `SERVER-CLASS`. The server can terminate its role as server for all class, server and service names for which it is registered, using a single `DEREGISTER` command.
- `EOC`
  The function `EOC` is used by either partner to terminate one or more active conversations.
- `RECEIVE`
  The function `RECEIVE` is used by the server to obtain new requests from a client, and in the case of conversations, to obtain subsequent related messages from the same client. This function is also used by clients that issue asynchronous requests and wish to obtain the server's reply at a later time. The field `CONV-ID` defines the behavior of this function. `RECEIVE,CONV-ID=NEW` signals the server's readiness to obtain the next available new request, whereas the value `CONV-ID=`*`nnn`* indicates that the next message within an existing conversation is being requested by the server. The client uses `RECEIVE,CONV-ID=`*`nnn`* to obtain asynchronously a reply from the server for an existing conversation.
- `REGISTER`
  The function `REGISTER` is used by a component of an application to identify its intention to become a server and satisfy requests issued to the named `SERVER-CLASS`, `SERVER-CLASS` and `SERVER-CLASS`.
- `SEND`
  The function `SEND` is used by the client either to make a new request or to send subsequent related messages within a conversation. This function is also used by servers, after satisfying a request, or during the course of a conversation, to reply to the client. The field `CONV-ID` defines the behavior of this function. The client uses `SEND,CONV-ID=NEW` to initiate a new request and the value `CONV-ID=`*`nnn`* when sending subsequent related messages in a conversation. The server always uses `SEND,CONV-ID=`*`nnn`* when replying to a client, where *`nnn`* indicates the identity of the existing conversation. The same syntax is used for both conversational and non-conversational modes.

ACI Syntax

Function	Fields in EntireX Broker Control Block
DEREGISTER	API = 1 or higher , BROKER-ID = BROKER-ID , USER-ID = user_id [, TOKEN = token] , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | * [, OPTION = QUIESCE | IMMED]
EOC	API = 2 or higher , BROKER-ID = BROKER-ID , USER-ID = user_id [, TOKEN = token] [, OPTION = CANCEL] , CONV-ID = conv_id | ANY [, SERVER-CLASS = class_name] [, SERVER-NAME = server_name] [, SERVICE = service_name]
RECEIVE	API = 1 or higher , BROKER-ID = BROKER-ID , USER-ID = user_id [, TOKEN = token] , WAIT = n | YES | NO , CONV-ID = conv_id | NEW | OLD | ANY , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | *
REGISTER	API = 1 or higher , BROKER-ID = BROKER-ID , USER-ID = user_id [, TOKEN = token] , SERVER-CLASS = class_name , SERVER-NAME = server_name , SERVICE = service_name [, OPTION = ATTACH]
SEND	API = 1 or higher , BROKER-ID = BROKER-ID , USER-ID = user_id [, TOKEN = token] [, OPTION = DEFERRED] , WAIT = n | YES | NO , CONV-ID = conv_id | NEW , SERVER-CLASS = class_name

Function	Fields in EntireX Broker Control Block
	`, SERVER-NAME = server_name` `, SERVICE = service_name`

Key ACI Field Names

The following table lists key ACI field names for implementing applications that use the client/server communication model. The other fields are available to identify partner programs, specify buffer lengths, convey error codes, etc.

See *Broker ACI Fields* for all fields.

ACI Field Name	Explanation
`SERVER-CLASS`	A client uses these fields to identify the service that it requires. A server uses this to offer a service.
`CONV-ID`	Identifier to obtain and specify the conversation. Also used to determine communication mode: conversational or non-conversational. See ***Conversationality***.
`FUNCTION`	Function code for one of the verbs (see *Key Verbs for `FUNCTION` Field*).
`OPTION`	Indication of specific Broker behavior, depending on the function.
`WAIT`	Time value to specify blocking or non-blocking of the conversation. See *Blocked and Non-blocked Broker Calls*.

Key Verbs for FUNCTION Field

The following table lists the most important verbs for the `FUNCTION` field.

See *Broker ACI Functions* for a complete list of functions.

Verb	Description
`REGISTER`	Inform the EntireX Broker that a service is available.
`RECEIVE`	Retrieve request from partner.
`SEND`	Send reply to the partner.
`EOC`	Terminate one or more conversations.
`DEREGISTER`	Remove the availability of the service.

Implementation of Client and Server Components

This example implements a simple non-conversational server and the appropriate client. The server is able to receive a request from the client and send back a reply. See *Conversationality*.

The following EntireX Broker functions are used to implement the server component:

Function	Explanation
LOGON	Log on the application to EntireX Broker.
REGISTER	Inform EntireX Broker about the availability of a service.
RECEIVE	Retrieve request from partner.
SYNCPOINT	Commit the sending or acknowledgment receipts of a UOW and examine status.
SEND	Send reply to the partner.
DEREGISTER	Remove the availability of the service.
LOGOFF	Log off the application from EntireX Broker.

The program flow of the *client* component is:

```
LOGON USER-ID=user-id
SEND SERVER-CLASS=server-class,SERVER-NAME=server-name,SERVICE=service
LOGOFF USER-ID=user-id
```

The program flow of the *server* component is:

```
LOGON
REGISTER SERVER-CLASS=server-class,SERVER-NAME=server-name,SERVICE=service
repeat
     RECEIVE SERVER-CLASS=server-class,SERVER-NAME=server-name,SERVICE=service
     (individual request processing: reply to client for each message)
     SEND CONV-ID=n
end-repeat
DEREGISTER SERVER-CLASS=server-class,SERVER-NAME=server-name,SERVICE=service
LOGOFF
```

The example above illustrates the structure of a typical server program. It consists of a server registration and a loop with RECEIVE / SEND cycles. This RECEIVE / SEND loop is normally interrupted by shutdown messages from administration programs.

The appropriate client component needs three functions:

Function	Explanation
LOGON	Log on the application to EntireX Broker.
SEND	Send request to partner.
LOGOFF	Log off the application from EntireX Broker.

The service offered by the server above is used by issuing a SEND operation within the client component of the application.

Both server and client perform a LOGON as the first call and LOGOFF as the last call. This enables security checks and saves resources in EntireX Broker.

Blocked and Non-blocked Broker Calls

The application can use the EntireX Broker control block field WAIT to determine whether Broker will automatically generate a WAIT in order for the command to be received or satisfied by the partner application.

- Non-blocked Command: WAIT=NO
- Blocked Command: WAIT=YES or WAIT=n
- Examples: WAIT

Non-blocked Command: WAIT=NO

- SEND
  An application sends a message via Broker to a partner application. The caller does not wait for the partner application to RECEIVE the message or to process it. The application subsequently performs RECEIVE commands if it intends to retrieve messages from the partner. This technique is frequently used by server applications when replying to clients after satisfying their requests; it can also be used by client applications that do not want to wait for the request to be serviced, such as when using units of work (see *Writing Applications: Units of Work*).
- RECEIVE
  Allows an application to ask for a message to be returned from the partner application. If the partner application has not yet communicated any messages to Broker using the SEND command, an ACI response code is given to the application, indicating no messages are currently available either for the designated class/server/service or for the conversation (if an existing conversation was established). This technique can be used by both client and server application components, especially in a multithreading context, where more than one communication thread is being maintained, or when programming units of work (see *Writing Applications: Units of Work*).

Blocked Command: WAIT=YES or WAIT=n

- SEND
  An application sends a request via Broker to a partner application. The calling application is automatically put into a WAIT state until the partner application has performed a RECEIVE operation to obtain the request and then processes it before issuing a reply, using the SEND command. Unlike the case where WAIT=NO, an inherent RECEIVE is generated to return the partner's reply. This technique is used by client applications only.
- RECEIVE
  An application asks for a message to be returned from the partner application. The calling application is automatically put into a WAIT state until the partner application has provided the necessary message through issuing a SEND command. If no messages are available during the specified waiting time, an ACI response code is given to the application, indicating no messages were available for the designated class/server/service or for the conversation (if an existing conversation was established). This technique is frequently used by server applications when waiting for messages to arrive from clients; it can also be used by client applications if the SEND and RECEIVE commands are programmed separately.

Examples: WAIT

The EntireX Broker allows both server and client applications to specify a WAIT time with the SEND or RECEIVE function. WAIT is a field in the ACI control block (see *Broker ACI Fields*). If a WAIT time is specified, the application is suspended until a reply is received or the timeout value has elapsed. If a timeout occurs, the EntireX Broker returns an error code to the calling program. If no WAIT time is specified, the application continues processing and collects the reply later.

Server applications typically use the WAIT field with a RECEIVE function in order to wait for requests. WAIT is not typically used with server SEND functions, allowing the server to continue processing instead of waiting for a request. For example:

```
LOGON
REGISTER service
repeat
     RECEIVE,CONV-ID=NEW,WAIT=nS
     (individual processing)
     SEND,CONV-ID=n,WAIT=NO
end-repeat
DEREGISTER service
LOGOFF
```

Client applications use the WAIT field with a SEND function in non-conversational communication if they require a reply. Because the mode is non-conversational, no conversation ID is returned to the client. The client must therefore wait for the reply from the server.

```
LOGON
SEND,CONV-ID=NONE,WAIT=nS
LOGOFF
```

A `RECEIVE` function with no `WAIT` time can be used to check if requests or data/messages are available for processing. Control is returned to the caller even if no request or data/message is available to satisfy the caller's operation. Appropriate error codes are returned when nothing is available.

```
LOGON
RECEIVE,CONV-ID=n,WAIT=NO
LOGOFF
```

The application can use the EntireX Broker control block field `WAIT` in the following ways to determine whether Broker will automatically generate a `WAIT` in order for the command to be received or satisfied by the partner application.

Conversational and Non-conversational Mode

The mode of communication is always controlled by the component of the distributed application that initiates communication. In the client and server model, this is the client side. When starting a communication, the `CONV-ID` field of the ACI control block is used to signal the communication mode to the Broker as follows:

- `CONV-ID=NONE`
  Coded on the service-requesting side (client program), it denotes non-conversational mode. EntireX Broker assigns a unique conversation ID to the communication that the client does not need to know.
- `CONV-ID=NEW`
  Coded in the client program, it denotes conversational mode. The EntireX Broker assigns a unique conversation ID to the communication, which is retrieved by the server and client program. This conversation ID must be specified in subsequent calls by both sides to refer to this conversation, until the conversation is ended by either side.

The server always retrieves the unique conversation ID and uses it when sending back the reply to the client. If no reply is required in non-conversational mode, the server ignores the conversation ID.

Non-conversational Mode

When implementing a non-conversational communication, the `CONV-ID` field is used by the server as follows:

```
LOGON
REGISTER service
repeat
     RECEIVE,CONV-ID=NEW
     (individual processing)
     SEND,CONV-ID=n
end-repeat
DEREGISTER service
LOGOFF
```

The client's `SEND` function is supplemented as follows:

```
LOGON
SEND,CONV-ID=NONE
LOGOFF
```

Conversational Mode

When implementing conversational communication, the server uses the `CONV-ID` field as follows:

```
LOGON
REGISTER service
repeat
     RECEIVE,CONV-ID=NEW
    repeat
         (individual processing)
         SEND,CONV-ID=n
         RECEIVE,CONV-ID=n
    end-repeat until conversation ended
end-repeat
DEREGISTER service
LOGOFF
```

The conversation is ended when *Message Class 0003 - EntireX ACI - Conversation Ended* is received. See *Error Handling*.

The client's `SEND` function is supplemented as follows:

```
LOGON
SEND,CONV-ID=NEW
SEND,CONV-ID=n
SEND,CONV-ID=n
EOC,CONV-ID=n
LOGOFF
```

EOC Reason

The reason for an `EOC` might be of interest to the partner of the conversation. EntireX Broker enables you to define the `CANCEL` option for an `EOC` function to indicate an abortive end of conversation. You can also distinguish between a timeout and a regular `EOC` on the basis of the error number. The error class is always *Message Class 0003 - EntireX ACI - Conversation Ended*; the error number specifies the actual circumstances.

Managing Conversation Contexts

It is possible to program a server application to handle several clients simultaneously and thus many conversations in parallel. Such a server is also capable of providing several different services and this technique can be used to reduce the number of different server applications executing on your machine. This increases throughput without wasting resources on a new service replica. The following features make it easier to implement a server that supports multiple conversations:

- Conversation Status
- Conversation User Data
- Stored EOC

Conversation Status

The Broker ACI control block contains a field named `CONV-STAT`. This is filled by Broker after a `RECEIVE` command. The following values are possible:

Value	Description
NEW	This is a new conversation. If the server needs to allocate a user-specific area, for example, this can be done without a comparison being made against existing conversations.
NONE	This message is a conversationless message. It is probably not necessary to create a user context, since the next request of this user is completely independent of this one, which is a requirement of conversationless communication. The implementation of mixed servers (conversational and non-conversational) is easier if it is known whether a message is conversational or not.
OLD	The message belongs to an existing conversation. The server can refer to the conversation user data to find the partner context. See ***Conversation User Data***.

Conversation User Data

Servers capable of serving multiple clients simultaneously are either stateless (servicing non-conversational requests) or they have to store conversation-related data for each user. This conversation-related context data is typically stored by the server application in a dynamic memory area. When a message is received, the user context related to that conversation must be located. This can be done by implementing a mapping structure in the application that can be indexed by the conversation ID, which returns the related context data.

Additionally, conversation-related contexts can be maintained by the Broker on behalf of the server application using the `USER-DATA` field in the ACI control block. Broker remembers information stored in the `USER-DATA` field when executing the `SEND` command. This data is returned to the application on subsequent `RECEIVE` commands executed within the same conversation. Therefore, your application is able to store information in `USER-DATA` when executing `SEND` commands and retrieve it on `RECEIVE` commands. The data in `USER-DATA` is considered binary and is untouched by the Broker.

Note: The USER-DATA is never transmitted from client to server or vice versa. Both sides of a conversation can store different USER-DATA, and both sides always receive their own data.

This USER-DATA helps with context areas as follows. A server application encounters a new conversation with the CONV-STAT API field. The user area is created and, typically, a first application confirmation is sent back to the client. Along with this SEND function, the server specifies the pointer to the user context - or the index into a context array, or whatever is available - into the USER-DATA. Whenever another request/message comes from that client via this conversation, this pointer/index is returned to the application, and the server has the context of the client application immediately, without having to scan a list of known conversations. Example:

```
* example of State-ful server program which utilizes
* USER-DATA to maintain application specific context
* information between successive messages within
* conversations with clients.

REGISTER #SERVER-CLASS #SERVER-NAME #SERVICE

  DO FOREVER
    RECEIVE #CONV-ID=ANY
    DECIDE ON FIRST VALUE #ERROR-CODE
      /* ============================
      /* NICE RETURN CODE
      VALUE '0'
        DECIDE ON FIRST VALUE #CONV-ID
          /* ========================
          /* NEW CONVERSATION
          VALUE 'NEW'
            #REQUEST-IN = #RECEIVE-BUFFER
            ... PROCESS NEW REQUEST FROM CLIENT AND
                REPLY TO CLIENT ASKING BROKER TO REMEMBER
                ACCOUNT NUMBER SO CLIENT DOESN'T HAVE TO
                TRANSMIT THIS WITH EVERY MESSAGE
            #ACCOUNT-NR = REQUEST-IN.ACCOUNT-NR
            SEND #CONV-ID #SEND-DATA #USER-DATA
          /* ========================
          /* EXISTING CONVERSATION
          NONE VALUE
            /* NEXT MESSAGE IN CONVERSATION RECEIVED
            /* AND ACCOUNT NUMBER REMEMBERED BY BROKER
            #ACCOUNT-NR = #USER-DATA
            #REQUEST-IN = #RECEIVE-BUFFER
            ... DO SOME PROCESSING BASED ON REQUEST AND
                ACCOUNT NUMBER REMEMBERED BY BROKER FOR
                THIS CONVERSATION CONTEXT
            ... REPLY TO CLIENT AS APPROPRIATE AND
                END CONVERSATION SOONER OR LATER
            SEND #CONV-ID #SEND-DATA #USER-DATA
        END-DECIDE
      VALUE '00740074'                /* RECEIVE TIME-OUT
        ESCAPE BOTTOM
      NONE VALUE                      /* REAL BROKER ERROR
        ... DEAL WITH A REAL BROKER ERROR
    END-DECIDE
  DOEND /* END FOREVER LOOP

DEREGISTER
```

Stored EOC

Servers that handle multiple conversations in parallel normally have to maintain a user context related to every conversation as described above. However, this context is typically allocated dynamically, and is therefore released after the conversation has ended. Not knowing when a particular conversation has finished would result in orphan contexts. To avoid this, the Broker offers the `NOTIFY-EOC` option, which is a service-specific attribute defined in the *Broker Attributes*.

This means that the `EOC` notification, even for timed-out conversations, is kept until the server receives it. This is useful for servers serving multiple conversations, since they are always informed about the end of a particular conversation and can therefore release all internal resources of a particular user context.

Specification of `NOTIFY-EOC=YES` can consume substantial system resources; as a result, a shortage of conversations for a service may occur. To avoid this shortage, a server must issue `RECEIVE` requests not restricted to any conversation, which gives the Broker the chance to report timed-out conversations. This does not of course mean that only `RECEIVE` functions with `CONVERSATION-ID=ANY` are valid, but from time to time such an unrestricted `RECEIVE` function should be issued.

Delayed SEND Function

To allow maximum flexibility in communication, the EntireX Broker provides a simple means of delaying the delivery of messages: allowing delivery of related messages in one logical block. If, for some reason, the messages that belong to a block cannot all be sent, all the messages in the logical block can optionally be deleted.

The mechanisms by which the EntireX Broker does this are the `HOLD` option on the `SEND` function and the `UNDO` function. Messages sent with `HOLD` status are not delivered until a message without the `HOLD` option is sent on the same conversation.

Example

This example illustrates the logical program flow of a client program that sends several messages on the same conversation, making delivery of the messages dependent on some condition. If the logical block of messages cannot be delivered (triggering an error condition), all messages in the logical block already sent can be deleted:

```
SEND,CONV-ID=NEW,OPTION=HOLD
....                                /* individual processing
SEND,CONV-ID=n,WAIT=NO,OPTION=HOLD
....                                /* individual processing
SEND,CONV-ID=n,WAIT=NO,OPTION=HOLD
....                                /* individual processing
if <error> then                     /* error condition
    UNDO,CONV-ID=n,OPTION=HOLD
else
    SEND,CONV-ID=n,WAIT=NO
end-if
....                                /* individual processing
EOC
```

Timeout Parameters

- Timeout Behavior
- Types of Non-activity Time
- Recommendations
- Unit of Work Lifetime
- Unit of Work Status Lifetime

Timeout Behavior

EntireX Broker provides a number of timeout mechanisms that allow you to control `WAIT` times flexibly, optimize resource usage, and configure efficient communication.

- The `CLIENT-NONACT`, `SERVER-NONACT` and `CONV-NONACT` attributes are non-activity timeout parameters that can be specified independently of each other to govern the three elements involved in a conversation: the requesting client, the registered server, and the conversation that will exist between them.
- The `WAIT` field in the Broker ACI control block allows you to place the sending or receiving program in a `WAIT` state for a specified time to allow data or a reply to be received before control is passed to the calling program. Placing the program into a `WAIT` state during a Broker command is referred to as issuing a blocked command. A non-blocked command is executed if `WAIT=NO` is specified. See *Blocked and Non-blocked Broker Calls*.

There is interplay between the `WAIT` values of your `SEND` and `RECEIVE` calls and the settings of the non-activity parameters in the Broker attribute file. See the `WAIT` field.

Types of Non-activity Time

There is interplay between the non-activity times specified in the attribute file for the attributes `CLIENT-NONACT` and `SERVER-NONACT`, where an application component performs more than one of these roles. In this case, the maximum non-activity time associated with the user will take precedence.

Recommendations

The following recommendations apply to developing client and server applications:

- Make the Broker `WAIT` time used for blocked `SEND` / `RECEIVE` calls in the application (both servers and clients) adjustable. This means that `WAIT` values must be read as a startup parameter from a user-supplied INI or CFG file, or any other parameter data set or set of environment variables, depending on the platform in use.
- On the client side, avoid high values for the `WAIT` time, which may lead to communication problems.
- When the `WAIT` time is lower than `CONV-NONACT` attribute, the caller will receive 00740074 error messages. Since the lifetime of the conversation exceeds the `WAIT` time specified for the command, the application can retry with the Broker function `RECEIVE`, and option `LAST` is possible.
- When the `WAIT` time is higher than `CONV-NONACT` attribute, the caller will receive 00030003 error messages. Since the lifetime of the conversation is less than the `WAIT` time specified for the command, it is not possible for the application to retry because any messages relating to the current conversation have already been cleaned up.

See also *Timeout Considerations for EntireX Broker*.

Unit of Work Lifetime

The `UWTIME` parameter in the *Broker Attributes* specifies the lifetime for a persistent UOW. The UOW exists until it has been successfully processed or until it is explicitly cancelled or backed out. If a UOW times out before being processed, or before any other explicit action is taken, its status changes to `TIMEOUT`. The status may or may not be retained in the persistent store, depending on the value of UOW status lifetime as described below. The default UOW lifetime for the Broker is defined by the `UWTIME` attribute. It can be overridden by the application in the `UWTIME` field of the ACI control block.

The UOW lifetime for the units of work is calculated only while Broker is executing.

Unit of Work Status Lifetime

This can be specified through either of the following two exclusive attribute settings. The default value zero implies the UOW status lifetime is zero, which means the status of the `UOWSTATUS` is not retained after one of the following events occurs: UOW is processed; UOW times out; UOW is backed out; UOW is cancelled. Status lifetime can be specified through either of the following two parameters in the *Broker Attributes*:

- `UWSTATP` (ACI_VERSION 3 or above)

  This attribute contains a multiplier used to compute the lifetime of the status of a UOW. See *Writing Applications: Units of Work*. The `UWSTATP` value is multiplied by the `UWTIME` value (the lifetime of the associated UOW) to determine how much additional time the UOW status is retained in the persistent store. The lifetime is calculated to start when any of the above events occurs and ends when the lifetime value expires. It can be overridden by the application in the `UOW-STATUS-PERSIST` field in the ACI control block.

- `UWSTAT-LIFETIME` (ACI_VERSION 8 or above)

  This attribute specifies the value to be added to the `UWTIME` (lifetime of the associated `UOWSTATUS`) to compute the length of time the UOW status is persisted. The UOW status lifetime begins at the time at which the associated UOW enters any of the following statuses: `PROCESSED`, `TIMEOUT`, `BACKEDOUT`, `CANCELLED`, `DISCARDED`. Specifying unit of work status lifetime in this way excludes specifying it as a multiplier value through the attribute `UWSTATP`.

The status lifetime for the unit of work is calculated only while Broker is executing.

Note: The values described here as `UWSTATP` and `UWSTAT-LIFETIME` can also be assigned as global Broker attributes or as a per-service attribute. However, the value specified by the application in the ACI control block overrides the Broker (or service) attributes. See *Broker ACI Fields*.

Data Compression

Data compression within EntireX Broker allows you to exchange smaller packet sizes between senders and receivers. This helps to reduce response time during transmissions as well as improve the overall network throughput, especially with low bandwidth connections.

Compression is performed only on the buffers used to send and receive data. The application has the option of setting the level of compression/decompression for data transmission. The compression level can be set to achieve either no compression or a range of compression/decompression. See *Data Compression in EntireX Broker*. Application components can set compression individually to Broker.

zlib is a general-purpose software implementing data compression across a variety of platforms. The functions used within EntireX Broker represent a subset of those available within the zlib software. The compression algorithms are implemented through the open source software **zlib**. It may occur that the data buffer does not compress during a data transmission; if it does not compress, a logged warning message will appear in 00200450 and in the stub.

Technique

The Broker ACI control block contains a field that is used to set the compression level. This field determines for any send/receive transmission whether the data buffer will be compressed/decompressed. See ACI control block field `COMPRESSLEVEL`.

Error Handling

After every broker operation, the application must check the `ERROR-CODE`. It consists of a combination of

- error class (first four digits) and
- error number (last four digits)

While the error number describes the exact situation, the error class often determines how the program will proceed after returning from the EntireX Broker operation. From the programmer's point of view, therefore, the error class may be more important than the particular error number.

For more information, see *Error Messages and Codes*.

Programming Techniques

We recommend trapping the error classes in a “case” statement, for example, a `DECIDE` in Natural or a switch statement in C.

All error classes - for example user and configuration errors - leading to the same action (that is, reporting or logging the situation and aborting issuing broker calls), can be handled together in the `NONE VALUE` or default case.

Example for C Programming Language

```
int   i, iErrorCode, iErrorClass, iErrorNumber, ret_val;
char  szErrorTextBuffer[S_TXT + 1];.....

/* prepare error code field and error text buffer */
memset(pETBCB->error_code,'0',sizeof(pETBCB->error_code));
memset(szErrorTextBuffer,'\0',sizeof(szErrorTextBuffer));

/* call the broker */
ret_val = broker(pETBCB,pSendBuffer,pReceiveBuffer,szErrorTextBuffer);

/* evaluate error class from error code field */
iErrorClass = 0;
for(i = 0; i < 4; ++i)
{
    iErrorClass *= 10;
    iErrorClass += pETBCB->error_code[ i ] - '0';
}

if (iErrorClass == 0 && ret_val != 0)
{
    printf("Wrong API_TYPE and/or API_VERSION\n");
}
else
{
    /* evaluate error number from error code field */
    iErrorNumber = 0;
    for(i = 4; i < 8; ++i)
    {
        iErrorNumber *= 10;
        iErrorNumber += pETBCB->error_code[ i ] - '0';
    }

    /* evaluate error code as integer value */
    iErrorCode = (iErrorClass * 10000) + iErrorNumber;

    /* handle error */
    switch (iErrorClass)
    {
        case 0: /* Successful Response */
            ....
            break;

        case 2: /* User does not exist */
            ....
            break;

        case 3: /* Conversation ended */
            ....
            break;
```

```
        case 7: /* Service not registered */
            ....
            break;

        case 74: /* Wait Timeout occurred */
            ....
            break;

        ....

        default:
            printf("EntireX Broker Error occurred.\n");
            printf("%8.8u %s",iErrorCode,szErrorTextBuffer);
            break;
    }
}
```

Using Send and Receive Buffers

- Introduction
- Error Cases
- Transport Methods

Introduction

The send buffer and the receive buffer are passed as parameters to the EntireX Broker. Both buffers can occupy the same location. See *Call Format* in the language-specific EntireX ACI documentation.

The length of the data to be sent is given in the ACI field `SEND-LENGTH`. If the `SEND-LENGTH` is greater than the send buffer during data transmission, you could accidentally send the data that is physically located in memory behind your send buffer to the designated Broker.

The `RECEIVE-LENGTH` is required with the `RECEIVE` function and with `SEND` functions waiting for a reply. The length of the receive buffer is specified in the ACI field `RECEIVE-LENGTH`. If the `RECEIVE-LENGTH` is greater than the receive buffer during data reception, you can overwrite the data physically located behind the receive buffer being used.

If the data to be returned is less than `RECEIVE-LENGTH`, the rest of the receive buffer remains unchanged and is not padded with trailing blanks or other characters. The ACI field `RETURN-LENGTH` contains the length of the data actually returned. The `RECEIVE-LENGTH` field is not changed upon return.

Note: With Adabas version 8, the maximum size of message data is no longer limited to approximately 32 KB. If Adabas version 8 is not used, these same limits still apply under z/OS.

Error Cases

Character conversions of data can increase the amount of data and thus require a buffer of a larger size than provided. It may also be impossible to determine the size required in advance. EntireX provides a feature to reread the data in such cases:

Using API version 2 and above, if the amount of data to be returned is greater than the `RECEIVE-LENGTH`, the exact length needed is given in the ACI field `RETURN-LENGTH` together with an error code, depending on the character conversion approach. See *Internationalization with EntireX*. Note the following:

- For *Translation User Exit*:
  - The error code is 00200094.
  - The data up to the length of the receive buffer is translated. The rest is truncated.
- For *ICU Conversion* and *SAGTRPC User Exit*:
  - The error code is 00200377.
  - No data is returned in the receive buffer.

To obtain the entire message, increase the size of the receive buffer and issue an additional Broker ACI function `RECEIVE` with the option "LAST".

Using API version 5 and above, it is also possible for a client to reread a truncated message in non-conversational mode, by issuing an additional Broker ACI function `RECEIVE` with the option "LAST" as well as the `CONV-ID` returned from the ACI control block. No `EOC` is needed after `RECEIVE`.

Transport Methods

The maximum length possible for send and receive buffers is affected by the transport method used.

Transport Method	Maximum Receive / Send Buffer Size	If using this transport method, ...
TCP/IP	2,147,482,111 B	■ the maximum send and receive buffer size is approximately 2,147,482,111 bytes.
Entire Net-Work	30,545 B	■ the send and receive buffer sizes are affected by the setting of the Net-Work parameter `IUBL` for all involved platforms (see the Net-Work documentation for more information); ■ the send and receive buffer sizes are affected by the Adabas SVC/Entire Net-Work-specific attribute `IUBL` for Broker running under z/OS; ■ the maximum send and receive buffer size is around 30,545 bytes.

Transport Method	Maximum Receive / Send Buffer Size	If using this transport method, ...
		Note: Under z/OS with Adabas version 8, the value for NET is the same as for TCP.
SSL	2,147,482,111 B	■ the maximum send and receive buffer size is approximately 2,147,482,111 bytes.

Tracing

Trace information showing the commands help the application programmer debug applications and solve problems. Tracing can be obtained for the application (stub trace) and for the Broker kernel (kernel trace). The stub trace shows the Broker functions issued by your application, whereas the Broker kernel trace will contain all Broker functions issued by all applications using the Broker.

Setting the Broker attribute `TRACE-LEVEL=1` provides traces containing just the Broker functions processed by the Broker kernel without additional diagnostics. It is only necessary to set the trace value higher when generating traces for Software AG support.

Stub Trace

Tracing is available for all stubs on UNIX and Windows. For the stubs for which tracing is available on z/OS, see table under *Administering Broker Stubs*.

To set the stub trace, see *Tracing for Broker Stubs* in the platform-specific Administration documentation.

Kernel Trace

Tracing is available for Broker on all platforms. For z/OS, see *Administering Broker Stubs*.

To set the kernel trace, see *Tracing webMethods EntireX* in the platform-specific Administration documentation.

Transport Methods

- Overview of Supported Transports
- TCP/IP
- Entire Net-Work
- SSL/TLS
- Considerations for Writing Applications
- Restrictions with API Versions 1 and 2

Overview of Supported Transports

This table gives an overview of the transport methods supported by EntireX Broker stubs.

Operating System	Environment	Module	Transport to Broker			
			TCP	SSL	NET [1]	HTTP(S) [5]
z/OS	Batch, TSO, IMS (BMP)	BROKER	x	[2]	x	
	Com-plete	COMETB	x	[2]	x	
	CICS	CICSETB	x	[2]	x	
	IMS (MPP)	MPPETB	x	[2]	x	
	IDMS/DC [3]	IDMSETB	x	[2]		
	Natural	NATETB23	x	[2]	x	
	UNIX System Services	*EntireX Java ACI*	x	x		x
UNIX		broker.so	x	x		
		EntireX Java ACI	x	x		x
Windows		broker.dll [4]	x	x		
		EntireX Java ACI	x	x		x
BS2000	Batch, Dialog (formerly TIAM)	BROKER	x		x	
z/VSE	Batch	BKIMB	x	[6]	x	
	CICS	BKIMC	x	[6]	x	
IBM i		EXA	x			

Notes:

1. NET is available for transport to a broker running under mainframe platforms only; not to a broker running under UNIX or Windows.
2. Under z/OS, use IBM's Application Transparent Transport Layer Security (AT-TLS). Refer to the IBM documentation for more information. See also *SSL/TLS and Certificates with EntireX*.
3. Tracing and transport timeout are not supported in this environment.

4. Stub broker32.dll is supported for reasons of backward compatibility. The functionality is identical to broker.dll.
5. Via EntireX Broker HTTP(S) Agent; see *Settting up and Administering the EntireX Broker HTTP(S) Agent* in the UNIX and Windows Administration documentation.
6. Under z/VSE, use BSI's Automatic Transport Layer Security (ATLS). Refer to the *BSI SSL Installation, Programming and User's Guide*. See also *SSL/TLS and Certificates with EntireX*.

See also:

- *Transport Methods for Broker Stubs* in the platform-specific broker stub Administration documentation
- *Setting Transport Methods* under *Writing Advanced Applications - EntireX Java ACI*

TCP/IP

TCP is not available for all Broker stubs and all environments (see table above).

See *Using TCP/IP as Transport Method for the Broker Stub* in *Transport Methods for Broker Stubs* in the platform-specific broker stub Administration documentation, which describes how to set up TCP transport.

Application programs using TCP/IP as the transport specify the target Broker ID in terms of a host name (or IP address) together with the port number on which the Broker TCP/IP communications driver is listening. Example: An application communicating through TCP/IP would specify on each command the Broker ID

```
IBM1:3932:TCP
```

where the host on which the Broker kernel executes is known to TCP as IBM1 and is listening on port 3932.

Entire Net-Work

Communication through Entire Net-Work is available for all Broker stubs when communicating with a Broker kernel on z/OS through Entire Net-Work. Applications can also utilize Entire Net-Work communication to obtain local interprocess communication with a z/OS Broker kernel running on the same machine as the application. This can provide a considerable performance benefit. Local interprocess communication is achieved through the Adabas SVC mechanism.

Application programs using Entire Net-Work as the transport specify the target Broker ID in terms of the target Entire Net-Work ID of the Broker kernel. For example, an application communicating through Entire Net-Work would specify on each command the Broker ID:

```
ETB001::NET
```

This can be abbreviated to the following for the Assembler stubs executing on z/OS (BROKER, CICSETB, COMETB, MPPETB):

```
ETB001
```

where the Entire Net-Work ID of the Broker kernel is 001.

SSL/TLS

Application programs using Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport must specify the SSL settings to the broker stub before any communication with the Broker can take place. There are various methods of setting SSL/TLS transport depending on programming language and platform. For ACI clients and ACI servers, see *Using the Broker ACI with SSL/TLS* (Assembler | C | COBOL | Java | PL/I).

Considerations for Writing Applications

- The ACI field `WAIT` allows the application to place the sending or receiving program in a `WAIT` state for a specified time; data or a reply will therefore be received before control is passed to the calling program. When a `WAIT` value is specified for a `SEND` / `RECEIVE` function, the calling application waits until the specified time has elapsed or a notification event occurs.
- `WAIT=YES` makes additional handling necessary in the Broker stub, whereby `YES` is replaced by the maximum integer value. We recommend you specify a finite value instead of `YES`.
- If frequent outages are expected in the network connections, it is useful to set the transport timeout to *n* seconds. After *n* seconds, the Broker stub terminates the TCP connection, if there is no response from the other side (the Broker kernel). This will help free up the network on the application side. In the case of applications for which the `WAIT` value is specified in the ACI control block (that is, blocking applications), the actual timeout value is the total of the transport timeout plus `WAIT` time.
- TCP/IP only:
  - The Broker ID can contain either an IP address or a hostname. If a hostname is used, it should be a valid entry in the domain name server.
  - A `LOGOFF` call to the Broker kernel will only logically disconnect the application from the Broker kernel. The physical TCP/IP connection is not released until the application terminates.

Restrictions with API Versions 1 and 2

The following maximum message sizes apply to all transport methods:

- ACI version 1: 32167 bytes
- ACI version 2: 31647 bytes

Variable-length Error Text

In previous ACI versions, Broker kernel always returned 40 bytes of error text, space-padded if necessary. For ACI version 9 and above, variable length error text can now be returned if requested. With ACI 9 and above, error text up to the requested length is returned via a new section in the ACI reply. For any previous ACI versions, ETXL is not sent, and the error text is returned by the traditional method.

Note that the error text will continue to be traced in the stub and kernel trace and kernel command log.

See *Broker ACI Fields*.

Programmatically Turning on Command Logging

You can trigger command logging for EntireX components that communicate with Broker by setting the field `LOG-COMMAND` in the ACI control block.

All functions with `LOG-COMMAND` programmatically set in the ACI string field will have their commands logged, regardless of any filter settings. Because the `LOG-COMMAND` option will override any command-log filter settings, remember to reset the `LOG-COMMAND` field if subsequent requests do not need to be logged.

4 Writing Applications: Units of Work

- What is a Unit of Work? 56
- Control Block Fields and Verbs 57
- Client/Server Programming for Units of Work 60
- Client/Server Programming for a Persistent Unit of Work 62
- Client/Server Restart after System Failure 64

This chapter describes the concept of units-of-work programming for EntireX Broker. Units of work are the precondition for achieving persistent messaging within your applications. Units of work can also be used without persistence.

This chapter assumes you are familiar with basic Broker ACI programming. If you are not familiar with it, we recommend beginning with the chapter *Writing Client and Server Applications*.

What is a Unit of Work?

A unit of work (UOW) is a group of related messages transmitted and received as a single entity. This is achieved through the sender committing as a single unit all the messages being sent and the receiver acknowledging receipt, as a single unit, of all the messages being received. Units of work are used in conjunction with conversations where a UOW exists strictly within one conversation. There can be more than one unit of work within a conversation. Where this is the case, subsequent UOWs can be created by either the client or the server. Since the conversation is always initiated by a client, the first UOW in the conversation is always created by the client. The UOW creator must commit the UOW to be created before being allowed to create another UOW within the same conversation.

Messages belonging to a UOW are always sent with `OPTION=SYNC`, or `OPTION=COMMIT`, which performs an implicit `COMMIT` at the same time as the `SEND`. Messages belonging to a UOW are always sent asynchronously, i.e. `SEND`,`WAIT=NO`. Messages belonging to a UOW are always received with `OPTION=SYNC` and can be received either with `WAIT=NO` or by specifying `WAIT`=[`YES | timevalue`], depending on application requirements.

Control Block Fields and Verbs

- Basic Functionality of Broker API
- ACI Syntax
- Key ACI Field Names
- Key Verbs for FUNCTION Field

Basic Functionality of Broker API

This section describes the expanded functionality of the Broker API used when programming units of work (UOWs) with or without persistence.

- `DEREGISTER`
  The function `DEREGISTER` is used by a server to indicate its intention to terminate its role as a server for the specified `SERVER-CLASS`, `SERVER-NAME` and `SERVICE`. The server can terminate its role as server for all class, server and service names for which it is registered, using a single `DEREGISTER`.
- `RECEIVE`
  The function `RECEIVE` is used by the server to obtain new requests from a client, and in the case of conversations, to obtain subsequent related messages from the same client. This function is also used by clients that issue asynchronous requests and wish to obtain the server's reply at a later time. The field `CONV-ID` defines the behavior of this function. `RECEIVE,CONV-ID=NEW` signals the server's readiness to obtain the next available new request, whereas the value `CONV-ID=`*`nnn`* indicates that the next message within an existing conversation is being requested by the server. The client uses `RECEIVE,CONV-ID=`*`nnn`* to obtain asynchronously a reply from the server for an existing conversation.
- `REGISTER`
  The function `REGISTER` is used by a component of an application to identify its intention to become a server and satisfy requests issued to the named `SERVER-CLASS`, `SERVER-NAME SERVICE`.
- `SEND`
  The function `SEND` is used by the client either to initiate a new conversation or to send subsequent messages within that conversation. This function is also used by servers to reply to the client during the course of a conversation. Each message is assigned to the unit of work currently being created by the sender. If this is the first message from the sender, a new UOW is created. Senders can create a subsequent unit of work by committing their existing UOW, creating and performing another subsequent `SEND` function. The field `CONV-ID` defines the behavior of this function regarding conversations. The client uses `SEND,CONV-ID=NEW` to initiate a new conversation and the value `CONV-ID=`*`nnn`* when sending subsequent related messages in a conversation. The server always uses `SEND,CONV-ID=`*`nnn`* when replying to a client, where *`nnn`* indicates the identity of the existing conversation. The `SEND` command is always used asynchronously with units of work, by both client and server. The sender can override the default persistence setting in the attribute file for the server class, server name and service, using the ACI field `STORE`.

- SYNCPOINT
  The function is used by either the client or the server when committing UOWs that they are creating, and also to acknowledge receipt of UOWs that they are receiving. It can also be used by the creator of a UOW to determine its current status or modify the status of a UOW at a later time.

ACI Syntax

Function	Fields in EntireX Broker Control Block
DEREGISTER	API = 1 or higher , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | * [,OPTION = QUIESCE | IMMED]
RECEIVE	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = SYNC , WAIT = n | YES | NO , CONV-ID = conv_id | NEW | OLD | ANY , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | * [,USTATUS = user_status] [,UOWID = uowid]
REGISTER	API = 1 or higher , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , SERVER-CLASS = class_name, , SERVER-NAME = server_name, , SERVICE = service_name
SEND	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = COMMIT | SYNC , WAIT = NO , CONV-ID = conv_id | NEW , SERVER-CLASS = class_name, , SERVER-NAME = server_name, , SERVICE = service_name

Function	Fields in EntireX Broker Control Block
	[,USTATUS = user_status] [,STORE = BROKER | OFF] [,UWTIME = uow_life_time] [,UWSTATUS-PERSIST = uow_status_persist_multiplier | UWSTAT-LIFETIME = uow_status_persist_lifetime] [,UOWID = uowid]
SYNCPOINT	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = BACKOUT | CANCEL | COMMIT | DELETE | EOCCANCEL | LAST | QUERY | SETUSTATUS [,CONV-ID = conv_id] [,UOWID = uowid] [,USTATUS = user_status]

Key ACI Field Names

ACI Field Name	Explanation
SERVER-CLASS	A client uses these fields to identify the service that it requires. A server uses this to offer a service.
CONV-ID	Identifier to obtain and specify the conversation. Also used to determine communication mode (non-conversational or conversational).
FUNCTION	Function code for one of the verbs; see ***Key Verbs for FUNCTION Field***.
OPTION	Indication of specific Broker behavior, depending on the function.
UOWID	Identifier generated by the Broker that identifies to the caller the unit of work ID. Specify valid UOWID to indicate an existing unit of work or leave blank when starting to SEND or RECEIVE a new unit of work. It is optionally specified when examining the status of a unit of work already created by the participant.
WAIT	Time value to specify blocking or non-blocking of the conversation. See *Blocked and Non-blocked Broker Calls*.

Key Verbs for FUNCTION Field

Verb	Description
`REGISTER`	Inform the broker that a service is available.
`RECEIVE`	Retrieve request from partner.
`SEND`	Send reply to the partner.
`DEREGISTER`	Remove the availability of the service.

Client/Server Programming for Units of Work

The figure below illustrates the logical program flow of a simple two-message client request UOW and a one-message server reply UOW. See also *Broker UOW Status Transition*.

1. The server logs on, registers, and issues a `RECEIVE` operation, and waits for a new CID and a UOW (unit of work).
2. The client logs on, creates a new UOW and a new conversation ID. It sends a message as part of a UOW and then commits the UOW, allowing the Broker to deliver it.
3. The server receives the first message in the UOW. Then the next (last) message. The server then creates a new UOW for the reply. The new UOW is part of the existing conversation (CID=123). The server commits both UOWs, i.e., the incoming UOW is processed and the outgoing UOW is ACCEPTED.
4. The client receives the incoming message and commits the UOW. The UOW is now PROCESSED.

Client	Server

	`LOGON,UID=,TOKEN=`
	`>OK`
	`REGISTER`
	`>OK`
	`RECEIVE,CID=NEW,OPT=SYNC,WAIT=1M`
	
	This receive operation will be satisfied by a new CID and a UOW. Non-UOW messages will not satisfy. (waits)

`LOGON,UID=,TOKEN=`
`>OK`
`SEND,OPT=SYNC,CID=NEW,WAIT=NO`
 	
Creates a new UOW and a new CID.	

Client	Server

`>OK,CID=123,UOWSTATUS=RECEIVED,`
`UOWID=987`
`SEND,OPT=SYNC,CID=123,WAIT=NO`	

Adds another message to the open UOW	

`>OK,CID=123,UOWSTATUS=RECEIVED,`
`UOWID=987`
`SYNCPOINT,OPT=COMMIT,CID=123`	

Commits the open UOW, allowing the broker to deliver it.	
`>OK,CID=123,UOWSTATUS=ACCEPTED,` `UOWID=987`	`>OK,CID=123,UOWSTATUS=FIRST,UOWID=987`
UOW (UOWID=987) is now safely in the hands of the broker.	The initial receive operation is completed, indicating a CID, a UOWID, and the FIRST message of a UOW.
`RECEIVE,CID=123,OPT=SYNC,WAIT=1M`	`RECEIVE ,CID=123,OPT=SYNC`
This will be satisfied by a UOW on CID=123.	Request the next message in open UOW.
(waits)	`>OK,CID=123,UOWSTATUS=LAST,UOWID=987`
	Receive the next message, which is the last. The server now has all the data.
	`SEND,OPT=SYNC,CID=123,WAIT=NO`
	Create a new UOW for the reply, on CID=123.
	`>OK,CID=123,UOWSTATUS=RECEIVED,UOWID=456`
	There are now actually 2 open UOWs (987 and 456), one in each direction.
	`SYNCPOINT,OPT=COMMIT,CID=123,` `UOWID=`
	This commits both UOWs, the incoming one (987) is now PROCESSED and the outgoing one (456) is ACCEPTED.
`>OK,CID=123,UOWSTATUS=ONLY,UOWID=456`	`>OK,CID=123,UOWSTATUS=ACCEPTED,` `UOWID=456`
Receive a message, the only one, in a UOW on CID=123. This is a different UOW than was sent.	

Client	Server
`SYNCPOINT,OPTION=COMMIT,CID=123`	(Loops back and reissues original receive)
This commits the UOW; it is now PROCESSED	

`>OK,CID=123,UOWSTATUS=PROCESSED,`
`UOWID=456`
`LOGOFF`
`>OK`	

Client/Server Programming for a Persistent Unit of Work

The figure below illustrates the logical program flow of a simple one-message persistent UOW with deferred delivery to a server, with no reply. The client queries the status of the UOW to determine its completion. See also *Broker UOW Status Transition*.

1. The client logs on and creates a new persistent UOW and a new conversation. The intended server is not currently available.
2. The client commits the open UOW, allowing the Broker to deliver it. The UOW (UOWID=987) is now stored by the Broker. It will be delivered whenever the server is available and will be retained even in case of system failure (that is, the UOW is persistent).
3. The client logs off.
4. The server logs on and registers. It receives the new conversation ID and the new UOW. The UOW is committed. Its status is now PROCESSED.
5. The client logs on using a user ID and token to identify itself as the client that originated the UOW. It then queries the status of its UOW. The status PROCESSED is returned, so the client knows that its UOW has been successfully delivered and processed by the server.

Client	Server

`LOGON, UID=,TOKEN=`
`>OK`
`SEND,OPT=SYNC,CID=NEW,WAIT=NO,`
`STORE=BROKER,`
`UWTIME=5M,UWSTATP=5`	

Creates a new persistent UOW and a new CID. The UOW will have a lifetime of 5 minutes; the duration of the status is 5 times this value (25 minutes). The intended server is not up at this time.	

Client	Server
`>OK,CID=123,UOWSTATUS=RECEIVED,UOWID=987` `SYNCPOINT,OPT=COMMIT,CID=123`	
Commit the open UOW, allowing the broker to deliver it.	
`>OK,CID=123,UOWSTATUS=ACCEPTED,UOWID=987`	
UOW (UOWID=987) is now safely in the hands of the broker. The UOW will be delivered whenever the server comes up, even if the system should fail.	
`LOGOFF`	
The client can now terminate, knowing that the UOW will be delivered.	
	Some time later, the server comes up.

	`LOGON,UID=,TOKEN=`
	`>OK`
	`REGISTER,`
	`>OK`
	`RECEIVE,CID=NEW,OPT=SYNC`

	This receive operation will be satisfied by a new CID and a UOW. Non-UOW messages will not satisfy.
	`>OK,CID=123,UOWSTATUS=ONLY,UOWID=987`
	The receive completes, indicating a CID and the ONLY message of a UOW.
	`SYNCPOINT,OPT=COMMIT,CID=123,` `UOWID=987`
	This commits the UOW; its status is now PROCESSED.
	`>OK,CID=123,UOWSTATUS=PROCESSED,` `UOWID=987`
	(Loop back and reissue original receive, if desired, or terminate)
Some time later, the client can come back and check the status of its UOW.	

Client	Server
`LOGON, UID=, TOKEN=`	
Specifying the same UID/TOKEN ensures that this client can be identified as the original client.	
`>OK`	
`SYNCPOINT,OPTION=LAST`	
Request the status of the last UOW this user created. The request must be made within 30 minutes, based on the value of the original SEND.	
`>OK,UOWID=987,CID=123,UOWSTATUS=PROCESSED`	
The client now knows that its UOW was successfully processed by the server.	
`LOGOFF` `>OK`	

Client/Server Restart after System Failure

Caution: `USER` and `TOKEN` must be specified when using persistent units of work (UOWs) to persist either a message or the status of a message exchanged between partner application components, where this information is held in the persistent store.

EntireX Broker provides a reconnection feature, using the `TOKEN` field in the ACI. If the application supplies a token along with `USER-ID`, the processing is automatically transferred when a request with the same user ID and token is received, either from the same process or from a different process or thread.

You need to specify `USER` and `TOKEN` to reconnect with the correct user context after a broker has been stopped and restarted when using units of work.

5 Writing Applications: Attach Server

- Implementing an Attach Server 66
- Implementing Servers started by an Attach Server 69

This chapter describes the programming of Attach Server for EntireX Broker. It assumes you are familiar with basic Broker ACI programming.

Implementing an Attach Server

An attach server is a server that is capable of starting another server rather than handling service requests itself. To implement an attach server, perform the following steps:

- Step 1: Register with EntireX Broker
- Step 2: Issue a Receive with Wait
- Step 3: Start Task
- Step 4: Deregister when the Work is Done

Step 1: Register with EntireX Broker

To register with EntireX Broker, the application has to add the `ATTACH` option to the `REGISTER` call. The `SERVER-CLASS`, `SERVER-NAME` and `SERVICE` parameters must reflect the service you can dynamically start. If the attach server is able to start several services, it has to register each service with the option `ATTACH` so that EntireX Broker knows exactly which services can be started by that attach server.

For example, an attach manager can start services (C1, N1, S1), (C2, N2, S2) and (C3, N3, S3). It therefore issues the following three registrations:

```
REGISTER  SERVER-CLASS=C1,SERVER-NAME=N1,SERVICE=S1,OPTION=ATTACH
REGISTER  SERVER-CLASS=C2,SERVER-NAME=N2,SERVICE=S2,OPTION=ATTACH
REGISTER  SERVER-CLASS=C3,SERVER-NAME=N3,SERVICE=S3,OPTION=ATTACH
```

Step 2: Issue a Receive with Wait

After all startable services have been registered by the attach server, the attach server must issue an unrestricted `RECEIVE` command in order to receive notification about queued service requests. The `RECEIVE` itself must be blocked for a certain time (`WAIT=`*nnn*). The attach server must be prepared to receive a notification for one of the announced services.

To continue the example from Step 1 above, the attach server now issues the `RECEIVE` command:

```
RECEIVE SERVER-CLASS=*,SERVER-NAME=*,SERVICE=*,WAIT=10M,RECEIVE-LENGTH=150
```

EntireX Broker answers either that no messages will be available after 10 minutes (error class 0074 is used for this kind of information) or that an attach service is required (error class 0010 and error code 0022), for example:

```
SERVER-CLASS=C2,SERVER-NAME=N2,SERVICE=S2,RETURN-LENGTH=116
```

with the following structure in the receive buffer, which is shown here in C programming language notation. The structure is the same for all programming languages and must be described in accordance with the programming language you select:

```
typedef struct
{
ETB_SHORT atm_version;   /*version of structure         */
ETB_SHORT atm_NotUsed;   /* alignment                   */
ETB_LONG  atm_nAttach;   /* # of failed server lookups   */
ETB_LONG  atm_nServer;   /* # of registered replicas     */
ETB_LONG  atm_nPendConv; /* # of pending conversations   */
ETB_LONG  atm_nActvConv; /* # of active conversations    */
ETB_CHAR  atm_server_class [S_SERVER_CLASS];/*class to attach        */
ETB_CHAR  atm_server_name  [S_SERVER_NAME]; /*server name to attach  */
ETB_CHAR  atm_service      [S_SERVICE];     /*service name to attach */
} ETB_ATMCB;
```

This structure contains the information necessary to decide whether a new replica needs to be started.

atm_nAttach

Number of client requests (`SEND CONVID=NEW`) the Broker could not schedule to a server immediately. After the Attach Manager has issued a `RECEIVE`, the value is reset to 0. If the Attach Manager does not issue its `RECEIVE`, this number shows the unreceived requests.

atm_nServer

Number of registered servers (replicas) minus those servers that are only finishing existing conversations (after issuing `DEREGISTER OPTION=QUIESCE`). This counter does not include the active Attach Server instances.

atm_nPendConv

Number of pending conversations, that is, client requests that could not currently be scheduled to a server. They are a subset of the active conversations.

atm_nActvConv

Number of the active conversations requesting a particular service.

Step 3: Start Task

This step depends very much on the platform. The attach server determines how to start up the desired application. The attach server only gets the logical name of the service. The mapping from the logical name to the program, including the path, startup parameters etc., must be performed by the attach server.

Step 4: Deregister when the Work is Done

Generally, attach servers are designed to "run forever". Once they are deregistered, no more services can be started on that platform automatically. However, if the administrator decides to shut down an attach server for whatever reason, he or she must `DEREGISTER` all registered services. There is no special flag for the deregistration.

After the final deregister, the attach server should perform a `LOGOFF` call to release all allocated resources:

```
DEREGISTER  SERVER-CLASS=C1,SERVER-NAME=N1,SERVICE=S1
DEREGISTER  SERVER-CLASS=C2,SERVER-NAME=N2,SERVICE=S2
DEREGISTER  SERVER-CLASS=C3,SERVER-NAME=N3,SERVICE=S3
```

or better

```
DEREGISTER SERVER-CLASS=*,SERVER-NAME=*,SERVICE=*
```

and as the last EntireX Broker-related command:

```
LOGOFF
```

Implementing Servers started by an Attach Server

In general, every server that can be used as a standalone server can be started up automatically. However, servers started by an attach server do not usually deregister and quit when no longer busy. They are not scalable, i.e. the number of replicas increases if not enough power is available, but the number does not decrease when there is no more work to be done.

To get around this situation, servers need to be prepared in such a way that they are started up automatically. Note the following points:

Notes:

1. The easiest server you can implement handles only one client for one conversation. After the last `EOC`, you can `DEREGISTER` or, preferably, `LOGOFF` the application and exit.
2. If you write an application that is automatically controlled by an attach server, try to implement the startup and the first `RECEIVE` as soon as possible. In other words, perform the necessary initialization after the conversation request is received.
3. Receive only the first call with the option `NEW`. Receive all subsequent calls with receive functions that are restricted to the established conversation (either with the option `OLD`, or with explicit restriction to the established conversation).
4. If you want to implement a server that does not exit after the first conversation, observe point 3 above. After the conversation has finished, set up the next `RECEIVE` with the option `NEW`. With this mechanism, the number of servers started in parallel corresponds to the number of clients trying to access the service simultaneously. This feature adapts the number of servers for high load peaks.
5. If you want to reduce the number of servers when they are no longer needed, set a proper `RECEIVE` timeout if you want to accept a new conversation, and finish your server if you actually receive a timeout. Both mechanisms give you the chance to react to load changes in both directions (increasing load and decreasing load).
6. Starting up a server for only one conversation is a simple server scheme, but you have to balance the simplicity of the application against the performance degradation for automatic startup. We recommend you use purely automatic server startup for servers only when the conversation is expected to last a reasonable length of time.
   If this is not clear, or if you want to run servers with short conversations - or even conversationless servers - you should consider using the method described under 4 and 5 above.

6 Writing Applications: Command and Information Services

- Accessing the Services 72
- Security with Command and Information Services 77
- Examples of Command Service 79

EntireX Broker provides an API for Command and Information Services (CIS) that include the following: shutting down conversations, servers and services; switching trace on and off; retrieving information on clients; registering servers and services.

Before you begin to write an application, see *Broker Command and Information Services*.

This chapter describes how to use the Command and Information Services from a programmer's point of view.

Accessing the Services

EntireX Broker's Command and Information Services are implemented as internal services. The method for requesting these services is exactly the same as the method for requesting any other service. An application issues a `SEND` function with appropriate data, retrieves the response with the receive data of the `SEND` function and, in the case of the information service, with additional `RECEIVE` operations. The `RECEIVE` operations have to be repeated until the information service indicates the end of data with an `EOC` return message.

Command and Information Services define a protocol that must be followed by the application. This protocol defines the structures needed to indicate to the service which information is desired and to return this information to the application so that the information can be interpreted.

Basic Rules

Several basic rules for command as well as information services are described here.

- Field Values
- Structures

Field Values

All fields necessary for a SEND function must be provided. The following values for SERVER-CLASS and SERVER-NAME are used for CIS:

Value	Description
SERVER-CLASS=SAG	Value is always SAG (Software AG).
SERVER-NAME=ETBCIS	Value is always ETBCIS (EntireX Broker Command and Information Services).
SERVICE=INFO	Full information service. Specify this for the full information service. All clients, servers and conversations are listed. See *Writing Applications using EntireX Security*.
SERVICE=USER-INFO	Limited information service. Specify this for limited information service. Only the user's own resources are listed. See *Writing Applications using EntireX Security*.
SERVICE=CMD	Specify this for the command service.
SERVICE=PARTICIPANT-SHUTDOWN	Specify this for the participant shutdown functionality.
SERVICE=SECURITY-CMD	Specify this for the EntireX Security command service.

The services do not have to be defined in the broker attribute file. Nothing has to be started or configured. You can use the services immediately after starting the broker.

The request for a command service or an information service is specified within the SEND buffer; the response - if there is one - is returned in the RECEIVE buffer.

Structures

Structures are used to describe the request and to return information. The following structures are available:

Structure	Information Service	Command Service	Description
Information Request Structure	Input		Used by an application to specify an information service request.
Command Request Structure		Input	Used by an application to specify a command service request.
Common Header Structure for Response Data	Returned	Returned	Returned as the first structure in each block from both the information service and the command service.
Information Reply Structures	Optionally Returned		The object-specific information reply structures are used to return information about these object types: ■ BROKER ■ CLIENT ■ CMDLOG-FILTER ■ CONVERSATION ■ NET ■ PARTICIPANT ■ POOL-USAGE ■ PSF ■ PSFADA ■ PSFCTREE ■ PSFDIV ■ RESOURCE-USAGE ■ SECURITY ■ SERVER ■ SERVICE ■ SSL ■ STATISTICS ■ TCP ■ TRANSPORT ■ UOW-STATISTICS ■ USER ■ WORKER ■ WORKER-USAGE

Command and Information Services can be accessed from any environment from which EntireX Broker can be accessed. The structures for these services are available for the programming languages Assembler, C, Natural and COBOL.

Accessing Information Services

For an information service request, the send buffer contains the information request structure with selection criteria depending on the requested information. See *Information Request Structure*.

Examples of Selection Criteria

```
OBJECT-TYPE = SERVICE
```

will return a list of all services.

```
OBJECT-TYPE = CONV, USER-ID = HUGO, TOKEN = FRED
```

will return a list of all conversations belonging to user with `USER-ID` HUGO who specified `TOKEN`=FRED within Broker calls.

```
OBJECT-TYPE = CONV, CONV-ID = 0815
```

will return information about the one single conversation with ID 0815.

When the `SEND` request returns, the receive buffer contains parts or all of the return data, and the `CID` field contains a conversation ID.

The return data in the receive buffer includes the common header structure followed by a list of one or more object type structures. See *Common Header Structure for Response Data*. For each object for which information is returned, there is one information reply structure containing the information.

Send Buffer	*Information Request Structure*
Receive Buffer	*Common Header Structure for Response Data* [Information Reply Structures]

Tips

- The size of the common header structure depends on the CIS interface version used.
- Test the error code in the common header structure. See *Broker Command and Information Services Error Codes*.
- If the receive buffer is not large enough to contain all available information, the remaining information can be obtained with additional `RECEIVE` functions in the same conversation. `WAIT=NO` can be specified because the data is there and only has to be collected. When no more data is available, the `RECEIVE` returns an end of conversation (`EOC`) message.

- If the selection is not unique - that is, more than one occurrence is possible - the information service returns a list (array) of information reply structures of the requested type. The common header structure informs the application of the total number of objects and the number of objects accompanying the reply data.
- The protocol for an information service request is as follows:

```
CALL BROKER
 FUNCTION=SEND    // send data = information request
 Service=USER-INFO
 CID=NEW
 WAIT=YES          // receive data = information reply
/* work off retrieved data */
REPEAT
 CALL BROKER      // receive data=information reply
  FUNCTION=RECEIVE
  Service=USER-INFO
  CID=n
  WAIT=NO
 IF End of Conversation
    escape
 END-IF
 /* work off retrieved data */
LOOP
```

- The initial `SEND` must be issued with the following:
  - `WAIT=YES` for blocking send commands
  - `CID=NEW` because the information service is implemented as a conversational service

Accessing Command Service

For a command service request, the send buffer contains the command request structure. See *Command Request Structure*. When sending a command service request, note the possible combinations under *Command Request Parameter Combinations*.

The return data in the receive buffer includes the common header structure (see *Common Header Structure for Response Data*):

Send Buffer	*Command Request Structure*
Receive Buffer	*Common Header Structure for Response Data*

Tips

- The error code in the common header structure must be tested by the application programmer. See *Broker Command and Information Services Error Codes*.
- A typical command service request looks like this:

```
CALL BROKER
  FUNCTION=SEND    // send data = command request
  Service=CMD
  CID=NONE
  WAIT=YES
```

- Unlike information service requests, the command service is defined as a non-conversational service that returns a single response. Therefore, the initial `SEND` must be issued with the following:
  - `CID=NONE`
  - `WAIT=YES`

Security with Command and Information Services

For security purposes, the Command and Information services are treated exactly like any other service. Therefore, if you are using either EntireX Security or equivalent user-written exits, user access to operate these services can be protected through your security system. This allows you to grant access based upon user ID to only those users who are authorized, where this facility is provided by the platform security implementation for Broker kernel.

- Full Command and Information Services
- Limited Information Services
- Protecting Specific Options

Full Command and Information Services

When using EntireX Security (or an equivalent), the full command service and the full information service are protected to avoid unauthorized access to information or potential disruption to systems. Therefore, you must grant appropriate access to the following resource profiles protecting the internal services:

- **Full Command Service**

```
Class: SAG Server: ETBCIS Service: CMD
```

- **Full Information Service**

```
Class: SAG Server: ETBCIS Service: INFO
```

Limited Information Services

The limited information service only returns information that belongs solely to the application making the request; it is not necessary to protect this service from unauthorized users. You can provide either limited or unlimited access to the resource profile used to protect the limited information service, as required:

- **Limited Information Service**

```
Class: SAG Server: ETBCIS Service: USER-INFO
```

Protecting Specific Options

The full command service can be used to shut down individual servers and, therefore, terminate any Class/Server/Service registered to the server application. When using EntireX Security (or equivalent), the shut-server operation is protected to avoid unauthorized termination of applications. This security check honors the Class/Server/Service of the server application. Therefore, you must grant appropriate access to resource profiles protecting the server application, which gives authorized users permission to register. This is in addition to the authorization for the full command service:

- **Full Command Service (Shut Service option)**

```
Class: ACLASS Server: ASERVER Service: ASERVICE
```

The full command service can be used as a `PARTICIPANT-SHUTDOWN` for individual participants currently active in the memory of the Broker kernel. When using EntireX Security (or an equivalent), the stop-participant operation is protected to avoid unauthorized use and potential disruption of systems. Therefore, you must grant appropriate access to the following resource profile:

- **Full Command Service (PARTICIPANT-SHUTDOWN option)**

```
Class: SAG Server: ETBCIS Service: PARTICIPANT-SHUTDOWN
```

The full command service can be used to administer EntireX Security. Currently the EntireX Security commands:

- allow the EntireX Security trace level to be changed independently of the Broker trace level
- allow all cached security information for a user to be cleared.

Therefore, you must grant appropriate access to the following resource profile:

- **Full Command Service (SECURITY-CMD option)**

```
Class: SAG Server: ETBCIS Service: SECURITY-CMD
```

The CIS commands `SHUTDOWN CONVERSATION` and `SHUTDOWN SERVICE` require the authorization to use the specified Class/Server/Service triplet and to use CIS commands.

See *Overview of EntireX Security* and *Writing Security Exits*.

Examples of Command Service

Example 1: ALLOW-NEWUOWMSGS

The Broker was restarted with the attribute `NEW-UOW-MESSAGES=NO`. This action will allow only consumption of UOWs to occur after Broker restart. Therefore, after the persistent store capacity has decreased to an acceptable level, the Broker administrator can issue the CIS command to allow new UOW messages in the broker. See `ALLOW-NEWUOWMSGS`.

Example 2: FORBID-NEWUOWMSGS

The Broker has been executing for a period of time when the Broker administrator notices that the persistent store is nearly at capacity. As a preventive action, the Broker administrator can issue the CIS command to forbid new UOW messages. See `FORBID-NEWUOWMSGS`. This action will cause only consumption of UOWs to occur in the Broker. Thereafter, when the persistent store capacity has been reduced to an acceptable level, the Broker administrator can issue the CIS command to allow new UOW messages in the Broker. See `ALLOW-NEWUOWMSGS`.

7 Writing Applications using EntireX Security

- General Programming Considerations 82
- Authentication 84
- Authorization 87

This chapter provides programming aids relevant to EntireX Security programming. It assumes you are familiar with the basics of EntireX Broker ACI programming. See *EntireX Broker ACI Programming*.

General Programming Considerations

See *Overview of EntireX Security* for overview of concepts and installation.

- ACI Versions and Security
- Is Broker Kernel Secure?

ACI Versions and Security

If your applications are using ACI versions 1 to 7, you will decide at installation time whether they are to communicate with a secured Broker. Your administrator will probably have installed components of EntireX Security into the Broker stub environment(s) and into the Broker kernel.

If your environment is configured using components of EntireX Security, your applications can communicate only with secured Broker kernels. If you attempt to communicate with both secured and non-secured Broker kernels, you will receive ACI response code 00200379, indicating “inconsistent security installation”.

To achieve greater flexibility, particularly when migrating applications from development to production, ACI version 8 introduces the new functionality described in the following table. For ACI version 8 and above, the application may assign to the broker control block field `KERNELSECURITY` one of the following values:

Value	Description
N	Application does not intend to communicate with a secured Broker kernel.
Y	Application intends to communicate with a Broker kernel which is secured using EntireX Security.
U	Application intends to communicate with a Broker kernel which is secured with the customer's own security exits.

This information indicates the application's intention and ensures that the correct execution occurs in the Broker stub and the Broker kernel. If the stub and the field `KERNELSECURITY` do not match, the application will receive ACI response code 00200379. If an improper value is assigned, it is treated as a blank. To make this assignment seamless, use an initial `KERNELVERS` command when communicating with each Broker kernel so that the field is assigned automatically.

Note: The default value (binary zero or space) specified in this field will result in the behavior being determined by the security configuration rather than programmatically. It is therefore possible to communicate either with a secure or non-secure Broker.

Is Broker Kernel Secure?

Issuing a KERNELVERS command will return information in the KERNELSECURITY field of the broker control block structure to indicate whether the application is communicating with a secure or non-secure Broker Kernel. This information can be important for ensuring the security of transactions and when making decisions such as prompting for USER-ID and PASSWORD values.

The following values are returned in the KERNELSECURITY field for ACI version 8 and above:

Value	Description
N	This is not a secured Broker kernel.
Y	This is a secured Broker kernel which is using EntireX Security.
U	This is a secured Broker kernel which is using the customer's own written security exits.

By issuing a KERNELVERS command, the appropriate value of KERNELSECURITY is automatically assigned to the control block structure; the user application does not need to take any further action other than supplying the correct USER-ID and PASSWORD. The application must maintain the contents of the control block structure for the duration of communication with the Broker kernel in order to retain the correct value of the KERNELSECURITY field. See *Broker ACI Control Block Layout* in the EntireX language-specific ACI documentation.

Notes:

1. Only applications using ACI version 7 or above can determine whether Broker is executing with security. In version 8 or above, the necessary information is automatically set up in the Broker control block.
2. We strongly recommend that applications maintain a separate copy of the Broker control block for each user ID (or USER-ID and TOKEN if specified). Furthermore, if the application communicates with different Broker kernels, a separate copy of the Broker control block must be maintained for each user and each Broker ID.

Authentication

- Authentication with User ID and Password
- Role of Security Token (STOKEN) during Authentication
- Trusted User ID
- Client User ID
- FORCE-LOGON

Authentication with User ID and Password

The application is responsible for assigning the correct `USER-ID` and `PASSWORD` values to the control block structure. This information is normally communicated through the `LOGON` command, since this command initiates the user's session with the Broker kernel. Where the attribute file contains `AUTOLOGON=YES` the first command issued by a user does not have to be `LOGON`, in which case the application must supply `USER-ID` and `PASSWORD` credentials for the commands `SEND` or `REGISTER`.

Supplying the `USER-ID` and `PASSWORD` could subsequently be required if the user times out due to expiration of either `CLIENT-NONACT` or `SERVER-NONACT` time limits. If the user context has timed out due to these inactivity limits being exceeded, one of the following events will occur when the application attempts to issue the next command.

Reason for ACI Error Action

- 00200134

  Application must perform another explicit `LOGON` with correct credentials in the `USER-ID` and `PASSWORD` fields:

  `AUTOLOGON=NO` in the attribute file, or `AUTOLOGON=YES` and `FORCE-LOGON=YES`.

- 00080003

  Application must supply correct credentials in `USER-ID` and `PASSWORD` fields:

  `AUTOLOGON=YES` in attribute file, `FORCE-LOGON=YES` not specified in the control block.

  Subsequent commands do not require explicit `LOGON` to be issued.

- 00080352

  Application has attempted to transfer control to a different thread, or process, without correctly transferring the necessary values of `USER-ID`, `TOKEN` and `STOKEN`:

  The application transferring control must make values of `USER`, `TOKEN` and `STOKEN` available to the application that is delegated to continue thread of execution.

- 00080353

  Application has not correctly maintained the value of security token (STOKEN) in the control block structure:

  The application must maintain the value of STOKEN in order to communicate securely with Broker kernel without sending PASSWORD with each command.

The application is able to change the password by assigning both PASSWORD and NEWPASSWORD fields of the control block structure. This must be done at the time of initial authentication or at a subsequent time when authentication is repeated due to timeout. It cannot be done at an arbitrary time by assigning the NEWPASSWORD field.

The PASSWORD and NEWPASSWORD fields are always communicated in an encrypted format.

Role of Security Token (STOKEN) during Authentication

EntireX Security automatically generates a non-repeated security token, which is placed in the ACI control block of the calling application. A unique security token is generated on behalf of all Broker participants only after successful authentication has occurred, and is used to ensure nobody can “tap in” to a participant's session. The calling application is responsible for maintaining the contents of the control block structure for the duration of its communication with the Broker kernel in order to ensure the correct value of security token is available on subsequent commands. An incorrect value of security token will cause access to be denied. Security token avoids the need for applications to supply a password except for presenting this once during the LOGON command, or the first command (excluding KERNELVERSION), if AUTOLOGON=YES is defined. If a LOGOFF command is issued or a participant is timed out, the password must be reentered so that a new unique security token can be generated.

An additional benefit of the security token is that it enables an application to transfer its execution to a different thread or even to a different process. This requires the application to make available the following fields of the control block structure to the program which is delegated to continue the thread of execution: USER, TOKEN and STOKEN. However, it is not necessary for the program transferring control to make its password available.

Note: If an application is unwilling or does not want to maintain the security token field (STOKEN) in the control block structure, it is possible for the systems administrator to configure the following field in the EntireX Security configuration module: BKISTK=Y. See *Ignore Security Token*.

Trusted User ID

This mechanism is available where at least one application and Broker kernel are executing on z/OS and communication is through Entire Net-Work (Adabas SVC).

Trusted User ID is an optional mechanism with which EntireX Security determines the identity under which the application is executing, without the application having to provide the `USER-ID` and `PASSWORD` in the Broker control block. See *Trusted User ID*.

The benefit of this mechanism is that application components executing on z/OS never have to provide credentials for authentication. This is because the identity under which execution occurs has already been verified when initially accessing the machine in each of these cases:

- online users
- batch jobs or started tasks.

All subsequent security authorization checks - for example `SEND` or `REGISTER` - are then performed under the known user ID under which the application executes.

Application components intending to utilize Trusted User ID must assign the Broker control block field `USER-ID` only. The value assigned to this field is arbitrary for security purposes but required in order to satisfy execution the stub. The application is not allowed to assign any value to `PASSWORD` if Trusted User ID is used. The following example is given:

```
USER-ID = 'SERVER123' /* arbitrary value: used by Broker but not
      significant for security purposes */
PASSWORD = ' ' /* password field must be
      set to blanks or binary zeros */
```

If the application does not clear the `PASSWORD` field, EntireX Security will assume that the application does not want to use Trusted User ID. Therefore valid credentials must be supplied to the `USER-ID` and `PASSWORD` fields in order to perform conventional authentication.

See also *Trusted User ID*.

Client User ID

Server applications are able to determine the user ID under which the partner client is executing by examining the content of the `CLIENT-USERID` field exposed in the Broker control block. Specifically, the `CLIENT-USERID` field should be examined on the first `RECEIVE` command of each new conversation to obtain the identity of the client. When EntireX Security is active, the server application is able to rely on the accuracy of the client user identity since it is derived from the user ID and password credentials supplied by the client.

See also *Trusted User ID* under *Writing Applications using EntireX Security* in the ACI Programming documentation and *Verified Client User ID* under *EntireX Security under z/OS* in the EntireX Security documentation.

FORCE-LOGON

`FORCE-LOGON` is used to override the `AUTOLOGON` feature of the Broker, with the result that the user does not log on to the Broker kernel implicitly with the first command issued but instead requires an explicit `LOGON`. When this option is used, it is necessary for the client and server to issue explicit `LOGON` function calls - even after the expiration of a client timeout `CLIENT-NONACT` or server timeout `SERVER-NONACT`. See *Timeout Parameters*.

`FORCE-LOGON` can be useful in cases where an implicit logon would be undesirable, for example when attempting to authenticate a user. Specifically, unless the password was communicated with every command, an implicit logon - after a period of inactivity - would fail because of a missing `PASSWORD`.

When `FORCE-LOGON` is set - and in the case of a client/server inactivity timeout - error 00200134 is returned instead of an implicit logon being performed automatically. Therefore, the specification of `FORCE-LOGON` can be used to give the programmer the opportunity to provide the ACI field `PASSWORD`, which is needed for successful authentication.

Note: Caution should be taken when repeating a failed authentication attempt for both an explicit and an implicit logon. Repeating the attempt several times can lead to a revocation of the user ID, depending on the configuration of your security system.

Authorization

- Client and Server
- Authorization Rules

Client and Server

Client applications are automatically subject to authorization requests if security is installed for EntireX Broker. For UNIX and Windows, see *Authorization Rules*.

An authorization check based on class, server and service is performed for the first `SEND` of a conversation and on every `SEND` if there is only one message in the conversation (`CONV-ID`). Messages are transmitted through to the server application only if the authorization check is successful; otherwise an ACI response is given to the client.

Server applications are automatically subject to authorization requests if security is installed for EntireX Broker. For UNIX and Windows, see *Authorization Rules*. An authorization check based on class/server/service is performed when the server application issues a `REGISTER` command. The server is allowed to register only if the authorization check is successful; otherwise an ACI response code is returned to the server application. In a similar way, if the administrator terminates a server through Command and Information Services (CIS), an authorization check is made based on the class/server/service.

The ACI error response codes encountered for authorization failures are: 00080009 | 00080010.

See also client-and-server example under *Resource Profiles in EntireX Security* and *Writing Client and Server Applications*.

Authorization Rules

An authorization rule is used to perform access checks for authenticated user IDs against lists of services defined within the rule. This feature is available on UNIX and Windows using EntireX Security on these platforms. Authorization rules can be stored in the Broker attribute file or in an LDAP repository. See *Authorization Rules* in the platform-independent administration documentation.

8 Broker ACI Fields

- Field Formats 90
- Field Descriptions 90

Field Formats

The ACI field formats are alphanumeric, binary, or integer and include the number of bytes. For example:

Format	Description
A8, A16, A32	Alphanumeric (A-Z, 0-9, underscore, hyphen). Other characters are currently possible, but we cannot guarantee that these will work consistently across all platforms in future versions. Do not use dollar, percent, period or comma.
B16, B32	Binary
Integer (unsigned)	

The terms “null value” or “nothing specified” used for a field mean blank for alphanumeric formats and zero for integer formats.

Field Descriptions

The ACI fields are described below in alphabetical order.

ACI Field	Format	Possible Values	API Vers	I/O	Description
`ADAPTER-ERROR`	A8		2	O	Filled by Broker with the transport error as supplemental diagnostic data.
`ADCOUNT`	I4		2	O	A count of the number of times an attempt was made to deliver a UOW. The count is incremented if a UOW is backed out or timed out.
`API-TYPE`	B1	bits	1	I	Required for all ACI functions. See *`API-TYPE` and `API-VERSION`*.
`API-VERSION`	I1	1-11	1	I	Required for all ACI functions.
`BROKER-ID`	A32	string	1	I	ID of the broker instance. Required for all ACI functions except `VERSION`. The `BROKER-ID` may be specified in URL-style or transport-method style. In order to communicate, applications must specify the same `BROKER-ID`. **Note:** URL style does not apply to mainframe platforms (z/OS, BS2000 and z/VSE).
`CLIENT-ID`	I4	1-2147483647	9	O	Returns to a server application the unique instance number of a client application.

ACI Field	Format	Possible Values	API Vers	I/O	Description
CLIENT-UID	A32	string	2	O	Applies only to client/server communication model. When a server issues a RECEIVE function, the user ID of the client is returned to the server in the CLIENT-UID field. If EntireX Security is installed, it is valid for the server application to rely on this user ID when making decisions concerning access to information. See *Authentication* (z/OS only). **Note:** There is an uppercase translation when the USER-ID field is propagated to the CLIENT-UID field under EntireX Security when Broker kernel is running under z/OS.
COMMITTIME	A17	YYYY MMDD HHMM SSms (millisecs.)	7	O	Time when UOW was committed.
COMPRESSLEVEL	A1	0-9 or Y | N	7	I	Compression level. See *Data Compression*. The following values are possible: **0 - 9**: 0 = no compression, 9 = maximum compression/decompression **N**: No compression **Y**: Compression level 6
CONV-ID	A16	string	1	I/O	A unique ID assigned to each conversation by EntireX Broker. Client and server must include the CONV-ID in their communications. Client and server can also specify the indicated textual values (capitals) in order to indicate to Broker the expected status of the conversation. Messages for the conversation are taken from the queue on a first-in, first-out basis. See *Conversational and Non-conversational Mode*. **NEW**: On a SEND function, initiates a new conversation. On a RECEIVE function, signals readiness to receive requests for new conversations only. A CONV-ID value is assigned to the conversation, and the value is returned to the caller.

ACI Field	Format	Possible Values	API Vers	I/O	Description	
					OLD	Applies to `RECEIVE` function only. Only messages for existing conversations are returned.
					ANY	On a `RECEIVE` function, requests or messages are returned on a first-in, first-out basis for any conversation. On an `EOC` function, any conversations belonging to the caller are terminated.
					NONE	On a `SEND` function, the message is non-conversational.
					conv-id	Indicates a specific conversation. The `CONV-ID` value is an internally generated identifier (containing numeric characters only or alphanumeric characters) for the conversation. Application programmers are advised to make no assumptions about the contents, layout, or meaning of any part of the `CONV-ID` field. If the client has specified `API-VERSION` 3 or above, the `CONV-ID` contains both alphanumeric and numeric characters. If the Broker does not support UOW processing (the Broker attribute `MAX-UOWS=0`) or the client has specified `API-VERSION` or `2`, the `CONV-ID` contains numeric characters.
`CONV-STAT`	I1	1 | 2 | 3	2	O	Conversation Status. See *Managing Conversation Contexts*.	
					1	`NEW` - The message is the first in a new conversation.
					2	`OLD` - The message is part of an existing conversation.
					3	`NONE` - The message is non-conversational.

ACI Field	Format	Possible Values	API Vers	I/O	Description
CORRELATION-ID	A64	string	11	O	Output value for function SEND if a message has been received. The CORRELATION-ID is the MESSAGE-ID that was used for the sent message. See *Unique Message ID* under *Broker ACI Functions* in the EntireX Broker ACI Programming documentation.
CREDENTIALS-TYPE	I1	0	9	O	Determines the credentials type to be used to authenticate a user. **0** Default. Use user ID and password.
DATA-ARCH	I1		4	I	Architecture code. For future use.
ENVIRONMENT	A32	string	1	I	Using the character conversion approach *Translation User Exit*, an ACI programmer can provide additional information to their translation exit through the ENVIRONMENT field, allowing flexible character conversion behavior in accordance with application requirements such as EBCDIC-ASCII conversion, byte swapping, and mixed data types. The ENVIRONMENT field can be used to pass this information from the application to the translation exit in Broker kernel. The field cannot be used for any other character conversion approaches and must be empty if a method other than translation user exit is used.
ERROR-CODE	A8		1	O	Returns an error code to the caller. The application should check the contents of this field at the completion of every Broker function. See *Error Handling*. The first four digits represent the error class; the next four digits represent the error number; see also *Error Messages and Codes*.
ERRTEXT-LENGTH	I4	0-40 | 0-255	1 | 9	I	Length of the error text buffer in bytes. See *Call Format* in the language-specific EntireX ACI documentation. If there are fewer than 40 bytes, the error text may be truncated. A value of 0 (zero) means no error text. **Note:** In previous ACI versions, Broker kernel always returned 40 bytes of error text that were space-padded if necessary. With ACI version 9 and above, variable-length error texts can be returned to improve logging and tracing.

ACI Field	Format	Possible Values	API Vers	I/O	Description
FORCE-LOGON	A1	Y | N	6	I	Override the AUTOLOGON feature of the Broker. See AUTOLOGON. **Y** — The attribute AUTOLOGON=YES in the Broker attribute file is overridden. See FORCE-LOGON under *Writing Applications using EntireX Security* in the ACI Programming documentation. **N** — Default. Use the value of the Broker attribute file for AUTOLOGON.
FUNCTION	I1	1-22	1	I	The Broker function to be performed. A function value is required and is modified by the ACI field OPTION and the other ACI fields. See below for description of values. **1** SEND, **9** LOGON **4** UNDO, **10** LOGOFF **5** EOC, **13** SYNCPOINT **6** REGISTER, **14** KERNELVERS **7** DEREGISTER, **16** SETSSLPARMS **8** VERSION, **22** REPLY_ERROR
KERNELSECURITY	A1	Y | U | N	7	I/O	This field is used by the application to indicate programmatically its intention to communicate with a secure/non-secure Broker. The field also indicates to the application how security has been configured for a particular Broker kernel. See Broker attribute SECURITY. When used as an **input** field, this field is used by programmer to indicate the desired security behavior of the application. If no option is specified, defaults to administrator's configuration setup. **Y** — EntireX Security **U** — User-written Security **N** — No security **Notes**

ACI Field	Format	Possible Values	API Vers	I/O	Description
					■ **Output** In version 7 or above, this field returns the output value when executing the `KERNELVERSION` command. ■ **Input** In version 8 or above, the application can programmatically specify the desired security behavior for all commands other than `KERNELVERSION`.
`LOCALE-STRING`	A40	string	4	I	The optional locale string contains a codepage name and tells the broker the encoding of the data. The application must ensure the encoding of the data matches the locale string. The broker stub itself does not convert your application data. The application's data is shipped and received as given. Under the Windows operating system: ■ The broker stub assumes by default the data is given in the encoding of the Windows ANSI codepage configured for your system. If you are using at least API-VERSION 8, a codepage identifier of this Windows ANSI codepage is automatically transferred to tell the broker how the data is encoded, even if no codepage name in the locale string is given. ■ If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation. ■ If you want to encode the data different to your Windows ANSI codepage, convert the data in the application and provide the codepage name in the locale string. During receive, decode the data accordingly. Under all other operating systems: ■ The broker stub does not automatically send a codepage identifier to the broker. ■ It is assumed the broker's locale string defaults match. See *Locale String Mapping*. If they do not match, provide the correct codepage name in the locale string here.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					Enable character conversion in the broker by setting the service-specific attribute `CONVERSION` to "`SAGTCHA`". See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. More information can be found under *Internationalization with EntireX*.
`LOG-COMMAND`	I1	0 | 1	9	I	Components that communicate with Broker can trigger command logging by setting this field. By default, command logging is based on the command log filters set in the kernel. You may override these kernel settings programmatically by setting this `LOG-COMMAND` field. If this field is set, all associated commands will be logged. **Note:** If command logging is not enabled for your kernel, you must first contact your administrator.

`LONG-BROKER-ID-LENGTH`	I4	0-2147483647	10		Length of `LONG-BROKER-ID`. If the value is non-zero, specify the value of `LONG-BROKER-ID` directly after the ACI control block. The `LONG-BROKER-ID` overrides any `BROKER-ID` value.
					
					With the `LONG-BROKER-ID` you can now specify numeric IPv6 addresses. Some sample values:
					
					`tcpip://[2001:0db8:85a3:08d3:1319:8a2e:0370:7347]:3930`
					
					`[2001:0db8:85a3:08d3:1319:8a2e:0370:7347]:3930:TCP`
					
					`(2001:0db8:85a3:08d3:1319:8a2e:0370:7347):3930:TCP`
					
					The IP address is enclosed in square brackets or parentheses.

`MESSAGE-ID`	A64	string	11	I/O	Input value for function `SEND` if `USE-SPECIFIED-MESSAGE-ID` is 1. The supplied `MESSAGE-ID` is used as unique identification for the message in the send buffer. Output value of function `GET-MESSAGE-ID`. If a message has been received, this field returns the corresponding `MESSAGE-ID`. Note that the sender must use API version 11 or above to receive a `MESSAGE-ID`.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					This `MESSAGE-ID` is a unique indicator for the corresponding message. See *Unique Message ID* under *Broker ACI Functions* in the EntireX Broker ACI Programming documentation.
`NEWPASSWORD`	B32	Can contain binary data.	2	I	Specifies a new password to be transmitted to the Broker kernel to check the authentication of the application. See *Authentication*. The current password can be changed only when the client or server authenticates itself. This occurs on the first Broker ACI function (can be `LOGON`) and requires the application to assign to the Broker ACI fields `PASSWORD` and `NEWPASSWORD`.
`OPTION`	I1	0-21	1	I	Provides additional information that modifies the behavior of the Broker ACI functions. **0** no option; **1** `MSG`; **2** `HOLD`; **3** `IMMED`; **4** `QUIESCE`; **5** `EOC`; **6** `CANCEL`; **7** `LAST`; **8** `NEXT`; **9** `PREVIEW`; **10** `COMMIT`; **11** `BACKOUT`; **12** `SYNC`; **13** `ATTACH`; **14** `DELETE`; **15** `EOCCANCEL`; **16** `QUERY`; **17** `SETUSTATUS`; **18** `ANY`; **19** reserved for future use; **20** `DURABLE` (deprecated); **21** `CHECKSERVICE`
`PASSWORD`	A32	Can contain binary data.	1	I	Specifies a password to be transmitted to the Broker to check the authentication of the application. See *Authentication* and `FORCE-LOGON` under *Writing Applications using EntireX Security* in the ACI Programming documentation.
`PTIME`	A8		2	I	Not used by EntireX Broker.
`RECEIVE-LENGTH`	B32	Binary.	1	I/O	Specifies the length of receive buffer, in bytes. The maximum length depends on the transport method: NET 30,545 TCP 2,147,483,647 SSL 2,147,483,647 **Note:** Under z/OS with Adabas version 8, the value for NET is the same as for TCP.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					See *Using Send and Receive Buffers*.
RETURN-LENGTH	I4		1	O	Length, in bytes, of the data returned. See *Using Send and Receive Buffers*.
SECURITY-TOKEN	B32	binary	1	I/O	The contents of this field depend heavily on the implementation of the security exits. This field is utilized by EntireX Security. The application must maintain SECURITY-TOKEN between commands and not change this value. We recommend that the application allocate a separate ACI control block for each user if it issues commands on behalf of more than one user. For objects executing inside Web servers, assigning a unique value, such as 'session ID', to the ACI TOKEN field is highly recommended to ensure uniqueness of user at same physical location. See *Ignore Security Token*. If EntireX Security is not implemented, and you choose to write your own security exits, you can transmit an initial value to your security exit as a credential that is used to calculate the actual security token. After an application's authenticity has been verified by the security exits, the SECURITY-TOKEN can be used to avoid additional authentication checks.
SEND-LENGTH	B32	binary	1	I/O	Specifies the length of data being sent, in bytes. The maximum length depends on the transport method: **NET** 30,545 **TCP** 2,147,483,647 **SSL** 2,147,483,647 **Note:** Under z/OS with Adabas version 8, the value for NET is the same as for TCP. See *Using Send and Receive Buffers*.
SERVER-CLASS SERVER-NAME SERVICE	A32 each	string, case-sensitive	1	I/O	A client uses these fields to identify the service that it requires. A server uses this field to offer a service. Using all three fields allows you to organize servers, making them easier to identify, monitor, and maintain. Servers can be organized into server-classes, with each server providing a number

ACI Field	Format	Possible Values	API Vers	I/O	Description
					of different services. Each service must be defined in the attribute file (see *Service-specific Attributes*). The service fields are required with `SEND`, `RECEIVE`, and `EOC` functions when `CONV-ID` is set to `NEW`, `OLD`, or `ANY`. When a `CONV-ID` is supplied, the service fields are ignored. `SERVICE=*` or `SERVER-NAME=*` can be used on a `RECEIVE` function to indicate all services within a specified server or all servers within a specified server class. **Note:** Server classes "SAG", "Entire", "Adabas", "Natural", "ETB", "RPC" and Broker are reserved for Software AG. Do not use them in your applications.
`STATUS`	I1		2	I/O	Not used by EntireX Broker.
`STORE`	I1	0 | 1 | 2	2	I/O	Persistence or non-persistence of a UOW. Used with the first `SEND` function for a UOW to specify whether the UOW is persistent or not. Once established, the persistence of a UOW cannot be altered. **0** none - Defaults to the value specified for the service. **1** OFF - The UOW is not persistent. **2** BROKER - The UOW is persistent.
`TOKEN`	A32	string, case-sensitive	1	I	Optionally identifies the caller and, when used, is required for all Broker ACI functions except `VERSION`. See *`USER-ID` and `TOKEN`*. **Caution:** `USER-ID` and `TOKEN` must be specified by all applications that use UOWs held in the persistent store.
`UOW-STATUS-PERSIST`		0 - 255	3	I	The value of the `UOW-STATUS-PERSIST` field is used as a multiplier to calculate the lifetime for the persistent status of a UOW. The value is multiplied by the value of the broker attribute `UWTIME`. The value 255 can be specified to indicate no persistent status.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					0 Means that the multiplier will have the same value as the `UWSTATP` Broker attribute. **255** Means that there will be no persistent status for UOWs. **1-254** Any number in this range is a valid multiplier.
`UOWID`	A16		3	I/O	A unique identifier for a UOW. The value is returned on the first `SEND` or `RECEIVE` command within a UOW; the value must be provided on all subsequent `SEND`, `RECEIVE` and `SYNCPOINT` commands related to the same UOW. Client and server can also specify the indicated textual value (capitals) in order to indicate to Broker the following: **BOTH** Since a server receives a UOW and replies with a different UOW, both UOWs can be committed or backed out by specifying `UOWID=BOTH` for the `SYNCPOINT` command. *`uowid`* The *`uowid`* must be supplied in subsequent `SEND`, `RECEIVE` and `SYNCPOINT` commands related to the same UOW.
`UOWSTATUS`	I1		3	O	Contains the status of a UOW. EntireX Broker returns the `UOWSTATUS` field to the calling application in order to provide information about the condition of the specified UOW. **1** `RECEIVED` - One or more messages have been sent as part of a UOW but the UOW has not yet been committed. **2** `ACCEPTED` - The UOW has been committed by the sender. **3** `DELIVERED` - The UOW is currently being received. **4** `BACKEDOUT` - The UOW has been backed out by the sender. **5** `PROCESSED` - The UOW has been received and the receiver has committed it.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					6 `CANCELLED` - The UOW has been cancelled by the receiver. **7** `TIMEOUT` - The UOW was not processed within the time allowed. **8** `DISCARDED` - The UOW was not persistent and its data was discarded as the result of a restart. With the exception of `DELIVERED`, all `UOWSTATUS` values are persistent. Persistent values are kept until they are explicitly deleted by the user or the time limit is exceeded. The lifetime of the `UOWSTATUS` value is determined by the broker attribute `UWSTATP`. `UOWSTATUS` values in the following table are returned on a `RECEIVE` function to indicate whether the message being transferred is part of a UOW and, if so, its sequence within the UOW: **0** `NONE` - The message is not part of a UOW. **9** `FIRST` - The message is the first message in a UOW. **10** `MIDDLE` - The message is neither the first nor the last in the UOW. **11** `LAST` - The message is the last message in the UOW. **12** `ONLY` - The message is the only message in the UOW.
`USE-SPECIFIED-CORRELATION-ID`	I1	0 | 1	11	I	Only used for function `SEND`. **0** Default. No `CORRELATION-ID` is sent to the broker. **1** The value in field `CORRELATION-ID` is sent with this request. See *Unique Message ID* under *Broker ACI Functions* in the EntireX Broker ACI Programming documentation.
`USE-SPECIFIED-MESSAGE-ID`	I1	0 | 1	11	I	Only used for function `SEND`.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					0 Default. A `MESSAGE-ID` must be generated for the message that is sent with this request. **1** The existing value in field `MESSAGE-ID` must be used for the message that is sent with this request. See *Unique Message ID* under *Broker ACI Functions* in the EntireX Broker ACI Programming documentation.
`USER-DATA`	B16	binary	2	I/O	Conversation User Data. See *Managing Conversation Contexts*.
`USER-ID`	A32	string, case-sensitive	1	I	Identifies the caller and is required for all Broker ACI functions except `VERSION`. See *`USER-ID` and `TOKEN`*.
`USTATUS`	A32	string	3	I/O	User-defined information about a unit of work (UOW). It can be transmitted on a `SEND`, `RECEIVE`, or `SYNCPOINT` function and is returned to applications that query the status of the UOW. To update the `USTATUS` field, use `OPTION=SETUSTATUS`.
`UWSTAT-LIFETIME`	A8	*n*S | *n*M | *n*H | *n*D	8	I	Add value for persistent status lifetime in client and server communication model. This field is used to calculate the lifetime of the UOW status. The value of this field determines how long the UOW status is to be retained in the persistent store after the UOW is processed or timed out if it is not processed. This is an alternative to specifying `UOW-STATUS-PERSIST`. ***n*S** The number of additional seconds the UOW status will exist. ***n*M** The number of additional minutes the UOW status will exist. ***n*H** The number of additional hours the UOW status will exist. ***n*D** The number of additional days the UOW status will exist.

ACI Field	Format	Possible Values	API Vers	I/O	Description
UWTIME	A8	*n*S | *n*M | *n*H | *n*D	3	I	The lifetime of a UOW. The UOW exists until its lifetime expires or it is explicitly cancelled or backed out with SYNCPOINT OPTION=CANCEL or SYNCPOINT OPTION=BACKOUT. If the UOW is not committed, backed out, or cancelled before its UWTIME expires, the UOW is discarded and its status becomes TIMEOUT. UWTIME is specified on the first SEND function for a UOW; it is not allowed on a RECEIVE function. **_n_S** — The number of seconds the UOW can exist. **_n_M** — The number of minutes the UOW can exist. **_n_H** — The number of hours the UOW can exist. **_n_D** — The number of days the UOW can exist.
VARLIST-OFFSET	I4	0-2147483647	10	I	For Software AG internal use only.
WAIT	A8	NO | YES | *n*S | *n*M | *n*H	1	I	When a WAIT value (other than NO) is specified on a SEND or RECEIVE function, the caller will wait for a reply until the message is received or the specified time limit has been reached. See *Blocked and Non-blocked Broker Calls*. **NO** — Default. No wait. Control is returned to the caller. **_n_S** — The number of seconds the caller will wait for a reply. **_n_M** — The number of minutes the caller will wait for a reply. **_n_H** — The number of hours the caller will wait for a reply. **YES** — Depending on the role of the user (client or server), the respective attribute is used (CLIENT-NONACT | SERVER-NONACT). If a server registers multiple services, the highest value of all the services registered is taken as wait time for the server. However, if the user is both client and server,

ACI Field	Format	Possible Values	API Vers	I/O	Description
					`CLIENT-NONACT` is also used for calculating the wait time. All different roles provide non-activity attributes. The maximum value is taken for the wait time.

9 Broker ACI Functions

- Overview Table 107
- Function Descriptions 108
- Option Descriptions 115
- ACI Field/Function Reference Table 117
- Unique Message ID 119

Programs written for EntireX Broker contain instructions that specify to the Broker which functions to perform. The function's behavior is controlled by the option value and other ACI fields.

See also *Broker ACI Fields*.

Overview Table

Function Name	Applicable Participant		Logon Required [1]	User ID Required [2]	Minimum API Version
	Client	Server			
DEREGISTER		X		X	1
EOC	X	X		X	1
GET-MESSAGE-ID	X	X		X	11
KERNELVERS	X	X		X	4
LOGOFF	X	X	X	X	2
LOGON	X	X		X	2
RECEIVE	X	X	[3]	X	1
REGISTER		X		X	1
REPLY_ERROR	X	X		X	8
SEND	X	X	[3]	X	1
SETSSLPARMS	X	X			6
SYNCPOINT	X	X	X[3]	X	3
UNDO	X	X	X	X	2
VERSION	X	X			2

Key

[1] Logon is a prerequisite for issuing this command. See LOGON.

[2] User ID is a prerequisite for issuing this command.

[3] The following functions require a logon when used with units of work: RECEIVE, SEND, SYNCPOINT.

Function Descriptions

DEREGISTER

This function is used by a server application to deregister a service from EntireX Broker. Assigned resources are deallocated. To remove multiple services, specify either `SERVER-CLASS`, `SERVER-NAME` and/or `SERVICE`.

Option	Description
`IMMED`	To execute a immediate deregistration, use `IMMED`. The service is removed immediately; an error code informs partners in existing conversations of this removal. Any active UOW is *backedout*.
`QUIESCE`	To execute a non-immediate deregistration, use `QUIESCE`. All active conversations are allowed to continue until an EOC is issued or a conversation timeout occurs. The application that issues the `DEREGISTER` function must remain active until all existing conversations are ended. No new conversations are accepted.

EOC

This function is used by a client or server and applies to conversational mode only. It is used to terminate one or more conversations. EntireX Broker accepts no additional `SEND`s for the conversation(s). The partner can receive requests and messages that were sent before the EOC was issued.

Although conversations are normally terminated by the client, the EOC function can be issued by either partner in a conversation. If an active UOW has not yet been committed (that is, its current status is *received* or *delivered*) the conversation will not be terminated until the UOW is either *committed*, *backedout*, *cancelled*, or *timedout*. See *Broker UOW Status Transition*.

- To terminate all conversations initiated by the participant, use `CONV-ID`.
- To terminate all conversations for a particular service, use `CONV-ID`, `SERVER-CLASS`, `SERVER-NAME` and/or `SERVICE`.

Option	Description
`CANCEL`	To inform the partner that the EOC is due to an unexpected event, use `CANCEL`.

GET-MESSAGE-ID

This function is used by clients and servers to generate a MESSAGE-ID for a subsequent SEND. GET-MESSAGE-ID makes it possible to access the MESSAGE-ID before sending it. API function SEND with field USE-SPECIFIED-MESSAGE-ID set to 1 is used to send the message with the previously generated MESSAGE-ID.

KERNELVERS

This function is used by any participant to determine the highest API-VERSION that is supported by the requested Broker. The highest API-VERSION that the Broker supports is returned in the API-VERSION field (see *API-TYPE and API-VERSION*). Platform and version information is returned in the error text.

Option	Description
CHECKSERVICE	If option is set to CHECKSERVICE, the command will determine whether a specified SERVICE is currently registered to the Broker.

The KERNELSECURITY field returns one of the following values to indicate whether the kernel is running with security. These values are returned only for API version 7 or above.

Value	Description
Y	Software AG-supplied security (SECURITY=YES in the Broker attribute file).
U	User-written security (SECURITY=YES in the Broker attribute file).
N	SECURITY=NO in the Broker attribute file.

LOGOFF

This function is used by all application components before termination when no further Broker functions are to be issued.

LOGOFF should be issued after the application's last SEND, RECEIVE or DEREGISTER has been executed. It releases all resources used by the application immediately rather than waiting until they time out (see *Timeout Parameters*).

LOGON

This function is used by all application components so that the application can establish communication with a particular instance of the Broker kernel.

Allows the client or server application to logon to EntireX Broker, which allocates the necessary structures to handle the new participant. If EntireX Broker is running in a secure environment (with `SECURITY=YES` in the attribute file), `LOGON` performs the authentication process.

`LOGON` is normally the first function.

In addition to the `USER-ID`, the `LOGON` optionally transmits the `PASSWORD`, `NEWPASSWORD` and `SECURITY-TOKEN` to authenticate itself, provided `SECURITY=YES` is set in the broker attribute file.

RECEIVE

This function is used by clients to receive incoming messages and by servers to receive incoming requests.

- You can specify a `WAIT` time, causing the `RECEIVE` to wait for the request or message that satisfies the operation.
- The `RECEIVE-LENGTH` field is required. It specifies the maximum length of data the caller can receive. A receive buffer of at least this length must be provided. The actual length of the message received is returned in the `RETURN-LENGTH` field.

Option	Description	Note
`ANY`	Used with the `RECEIVE` function to indicate that the `RECEIVE` will be satisfied by any message, whether part of a UOW or not.	
`LAST`	To retrieve the last (most recent) message in a conversation, use `LAST`.	With this option, `WAIT` must be set to `NO` or not specified.
`MSG`	To indicate that the `RECEIVE` will be satisfied only by a message that is not part of a UOW, use `MSG`. See also *Broker UOW Status Transition*.	
`NEXT`	To retrieve the next unprocessed request or message in a conversation, use `NEXT`.	
`PREVIEW`	To retrieve the next unprocessed request or message in a conversation without deleting the previous message or moving the `READ` pointer, use `PREVIEW`, which excludes using units of work.	
`SYNC`	To receive only messages that are part of a UOW, use `SYNC`. See also *Broker UOW Status Transition*.	

REGISTER

This function is used by servers to inform EntireX Broker that a service is available. The Broker obtains information about the service from the *Broker Attributes*, creates the appropriate environment, and makes the participant available as the specified `SERVER-CLASS`, `SERVER-NAME` and `SERVICE`.

If `REGISTER` is the first call by a server when both `AUTOLOGON` and `SECURITY` are set to `YES` in the Broker attribute file, `USER-ID` and `PASSWORD` are required in order to authenticate and authorize the server. This is because an implicit `LOGON` is being performed.

The services being registered must be defined in the attribute file.

Option	Description
`ATTACH`	To register an attach service, use `ATTACH`. An attach service cannot be requested by a client. Its function is to make available a service that cannot otherwise be scheduled.

REPLY_ERROR

This function is used by clients or servers to send an error message to the partner of the conversation. The error number is specified in the error code field. The sent message is delivered as an error text; the specified error number is delivered as an error code.

- The user must be logged on.
- The error number is a numeric 8-byte value and must start with 8, for example 80010001. A zero error number will be rejected. These errors are user-definable and therefore not documented.
- The error message is provided in the send buffer and is limited to 40 bytes.
- Use the `SEND-LENGTH` field to specify the length of the error message.
- `REPLY_ERROR` can be used with a valid `CONV-ID` only.
- Only `WAIT=NO` is allowed.
- The conversation is not allowed to contain units of work.

Option	Description
`EOC`	To end the conversation after the `REPLY_ERROR` function, use `EOC`.

SEND

This function is used by clients to send requests and by servers to send replies (messages). If a corresponding RECEIVE function issued by a partner application is outstanding, EntireX Broker forwards the request or message to that partner application. If not, EntireX Broker queues the request or message until a suitable RECEIVE is issued by a partner application. If no suitable RECEIVE is issued by a partner application, the request will timeout within the specified timeout period.

- You can specify a SEND with either of the following:
  - WAIT=YES | Value

    This causes an implicit RECEIVE to be generated and the SEND to wait for a reply. If a reply is expected, the SEND must pass the length of the receive buffer, in bytes, as the value of the RECEIVE-LENGTH parameter. The actual - not the specified - length of the reply is returned to the sender as the RETURN-LENGTH value.
  - WAIT=NO

    Choose WAIT=NO if you are only forwarding a request or message.
- Use the SEND-LENGTH field to specify the length of the request or message being sent. The specified number of bytes is transferred, starting at the beginning of the send buffer.
- The client starts a new conversation, using CONV-ID=NEW.
- The client can specify non-conversational mode, using CONV-ID=NONE.
- Include the SERVER-CLASS, SERVER-NAME and SERVICE if this is a new conversation or a non-conversational request.
- If you add the ENVIRONMENT parameter, its value is passed to the translation routine for the service.
- To transmit conversation-related data to the sending application, use USER-DATA.

Option	Description	Note
COMMIT	Use COMMIT to indicate that the UOW being sent is complete and can now be delivered to the intended receiver, which can be either client or server.	
EOC	To end the conversation after the SEND, use EOC.	WAIT must be NO or not specified.
HOLD	To hold SEND data in a queue, use HOLD. The data is released by a SEND without a HOLD.	WAIT must be NO or not specified.
SYNC	Client and server can send a message as part of a unit of work (UOW), using SYNC.	WAIT must be NO or not specified.

SETSSLPARMS

This function is only available on platforms where the broker stub directly supports SSL/TLS transport. See *Transport: Broker Stubs and APIs* It is used by ACI clients and ACI servers to set the SSL parameters. ACI-based clients or servers are always SSL clients. The SSL parameters are specified in the send buffer, (second parameter of the Broker ACI call). These SSL parameters are used and communication is performed if the Secure Sockets Layer is configured. See *SSL/TLS and Certificates with EntireX*.

To use SSL

1 See *Using the Broker ACI with SSL/TLS* (Assembler | C | COBOL | Java | PL/I).

2 Make sure the SSL server to which the ACI application (client or server) connects is prepared for SSL connections as well. The SSL server can be EntireX Broker or Broker SSL Agent. See:

- *Running Broker with SSL/TLS Transport* in the platform-specific Administration documentation
- *Settting up and Administering the EntireX Broker SSL Agent* in the UNIX and Windows Administration documentation

SYNCPOINT

This function allows you to manage units of work (UOWs), both persistent and non-persistent, that have been sent or received. See *Using Persistence and Units of Work*.

SYNCPOINT is used with the `OPTION` field. The `UOWID` parameter is required and normally limits the request to a specified UOW. For example:

```
SYNCPOINT OPTION=COMMIT,UOWID=n
SYNCPOINT OPTION=BACKOUT,UOWID=n
```

In cases where a server receives a UOW and sends a different UOW, you can ensure that the two UOWs will be processed together (that is, if one is committed, both are committed) by specifying `UOWID=BOTH`. For example:

```
SYNCPOINT OPTION=COMMIT,UOWID=BOTH
```

`UOWID=BOTH` can also be used with `BACKOUT`. This simply backs out both UOWs in a single call instead of two separate calls:

```
SYNCPOINT OPTION=BACKOUT,UOWID=BOTH
```

Option	Description
BACKOUT	Used by the sender, it causes the UOW to be deleted, with a status of *backedout*. By the receiver, causes the UOW to be returned to its prior, unreceived state, with a status of *accepted*. The ADCOUNT field is incremented. See also *Broker UOW Status Transition*.
CANCEL	Used by the receiver, it causes the UOW to be considered finished, with a status of *cancelled*. No further processing of the UOW is possible. The sender can cancel the UOW if, and only if, it is in *accepted* status. The following sequence of commands, issued during recovery processing, allows the sender to remove any created but undelivered UOWs: ■ SYNCPOINT OPTION=LAST ■ SYNCPOINT OPTION=CANCEL ■ SYNCPOINT OPTION=DELETE
COMMIT	User by the sender, it indicates that the UOW has been completely built and can be made available for delivery, with a status of *accepted*. By the receiver, indicates that the UOW has been completely received, with a status of *processed*. No further processing of the UOW is possible.
EOC	With UOWID=*n*, commits the UOW being created and ends the conversation.
EOCCANCEL	With UOWID=*n*, commits the UOW being created and cancels the conversation, that is, terminates the conversation immediately.
DELETE	With UOWID=*n*, deletes the persistent status of the specified UOW. The UOW must be logically complete (*processed*, *cancelled*, *timedout*, *backedout*, *discarded*) and must have been created by the caller.
LAST	Returns the status of the last committed UOW sent by the caller. In addition, SERVER-CLASS, SERVER-NAME, SERVICE and CONV-ID are also returned.
QUERY	With UOWID=*n*, returns the status of the specified UOW. In addition, SERVER-CLASS, SERVER-NAME and SERVICE details of the associated server are also returned.
SETUSTATUS	With UOWID=*n*, updates the user status of the specified UOW.

UNDO

This function is used to remove messages that have been sent but not received. It can only be used with an existing conversation. When a message is undone, the conversation continues.

Note: UNDO is not used in conjunction with units of work. See *Using Persistence and Units of Work*.

Option	Description
HOLD	To undo messages in HOLD status, use UNDO with HOLD.

VERSION

This function is used to return the version of the stub implementation in the receive buffer. This version string is useful to the application in determining the maximum API version supported by the stub and to Software AG support if problems occur.

The string was modified in version 8.2. Example:

```
EntireX Broker Stub XXXXXXXX Version=08.3.0.00, Highest API Supported=09
```

where "*XXXXXXXX*" is is the name of the stub, for example "CICSETB".

Option Descriptions

Number	Option	Used with Function	Description	Note
1	MSG	RECEIVE	Receive only a message that is not part of a UOW.	
2	HOLD	SEND	Places the messages in a HOLD queue. Messages are released by a SEND without HOLD.	Used in conversational mode only.
		UNDO	Releases all previously held messages.	
		LOGOFF	The conversation is not ended, even though the user is logged off.	
3	IMMED	DEREGISTER	Immediately terminate all conversations for the specified server. All partners are informed with an appropriate error code.	
4	QUIESCE	DEREGISTER	Terminate a server smoothly. Existing conversations are allowed to end normally; no new conversations are accepted. The server is removed from the “active” list.	Default option for this function.
5	EOC	SEND	End the conversation with the current message. It can be issued by either partner. The conversation is not ended if an active UOW has not yet been committed, that is, its status is *received* or *delivered*. See *Broker UOW Status Transition*.	
6	CANCEL	EOC	Abort a conversation rather than terminate normally. The receiver of a UOW can use	

Number	Option	Used with Function	Description	Note
			SYNCPOINT OPTION=CANCEL to interrupt processing and discard the UOW.	
7	LAST	RECEIVE	Used in conversational mode to retrieve the last (most recent) message.	
		SYNCPOINT	Returns the status of the last committed UOW sent by the caller.	
8	NEXT	RECEIVE	Retrieve the next unprocessed request or message. The request or message is then considered processed and can be accessed only with OPTION=LAST.	Default option for this function.
9	PREVIEW	RECEIVE	Retrieve the next unprocessed request message without deleting the previous message or moving the READ pointer. The previewed message will be retrieved again by the next RECEIVE OPTION=NEXT.	
10	COMMIT	SYNCPOINT	Commit the active UOW. ■ For a UOW being *sent*, it means that the UOW is complete and can now be delivered to the intended receiver. ■ For a UOW being *received*, it means that the UOW is complete and no further processing of the UOW is allowed.	
		SEND	Commit the active UOW.	
11	BACKOUT	SYNCPOINT	■ For *receiver* of a UOW: Return the UOW to its undelivered state. The UOW can be processed again, in its entirety, by subsequent RECEIVE operations. ■ For *sender* of a UOW: Delete the UOW. No further processing of the UOW is allowed.	
12	SYNC	SEND	Indicates that the message is part of a UOW.	
		RECEIVE	RECEIVE will be satisfied only by a message that is part of a UOW.	
13	ATTACH	REGISTER	Register an attach server.	
14	DELETE	SYNCPOINT	Delete the persistent status information for the specified UOW.	
15	EOCCANCEL	SYNCPOINT	Cancel the conversation after committing a UOW.	
16	QUERY	SYNCPOINT	Query the status of a UOW.	
17	SETUSTATUS	SYNCPOINT	Updates the user status of the specified UOW.	

Number	Option	Used with Function	Description	Note
18	ANY	RECEIVE	Indicate that the RECEIVE will be satisfied by a message that is or is not part of a UOW.	
19				No longer used.
20				No longer used.
21	CHECKSERVICE	KERNELVERS	Check if the specified service is active in EntireX Broker.	

ACI Field/Function Reference Table

The following table identifies the ACI fields that apply to each of the Broker functions. For a given function, an ACI field value may be a request field (Rq), and/or a reply field (Rt). Optional fields are marked (O).

ACI Field	Function													
	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	SETSSLPARMS	REPLY_ERROR	GET-MESSAGE-ID
ADCOUNT		Rt								Rt				
API-TYPE	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq
API-VERSION	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq Rt	Rq	Rq	Rq
BROKER-ID	Rq	Rq	Rq	Rq	Rq	Rq		Rq	Rq	Rq	Rq	Rq	Rq	Rq
CLIENT-UID		Rt												
COMMITTIME	Rt	Rt								Rt				
COMPRESSLEVEL	O Rt	O Rt											O	
CONV-ID	Rq	Rq	Rq	Rq						Rq Rt			Rq	
CONV-STAT		Rt												
CORRELATION-ID	O	O												
DATA-ARCH	O	O											O	
ENVIRONMENT	O	O											O	

ACI Field	Function													
	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	SETSSLPARMS	REPLY_ERROR	GET-MESSAGE-ID
ERROR-CODE	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt	Rt
ERRTEXT-LENGTH	O	O	O	O	O	O	O	O	O	O	O	O	O	O
FORCE-LOGON	O	O	O	O	O	O		O	O	O			O	
FUNCTION	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq	Rq
KERNELSECURITY											Rt			
LOG-COMMAND	Rq	Rq	Rq	Rq	Rq	Rq		Rq	Rq	Rq	Rq		Rq	
LOCALE-STRING	O	O											O	
MESSAGE-ID	O	O												Rt
NEWPASSWORD	O				O			O						
OPTION	O	O	O	O	O	O			O	O	O		O	
PASSWORD	O				O			O					O	
RECEIVE-LENGTH	O	Rq					Rq							
RETURN-LENGTH	Rt	Rt					Rt							
SECURITY-TOKEN	O	O	O	O	O	O	O	O	O	O			O	
SEND-LENGTH	Rq											Rq	Rq	
SERVER-CLASS	Rq	O		O	Rq	O				O Rt				
SERVER-NAME	Rq	O		O	Rq	O				O Rt				
SERVICE	Rq	O		O	Rq	O				O Rt				
STORE	O	Rt												
TOKEN	O	O	O	O	O	O		O	O				O	
UOWID	O Rt	O Rt								Rq				
UOWSTATUS	Rt	Rt		Rt						Rt				
USE-SPECIFIED-CORRELATION-ID	O													
USE-SPECIFIED-MESSAGE-ID	O													
UOW-STATUS-PERSIST										O				
USER-DATA	O	Rt								Rt			O	
USER-ID	Rq	Rq	Rq	Rq	Rq	Rq		Rq	Rq	Rq			Rq	Rq

ACI Field	Function													
	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	SETSSLPARMS	REPLY_ERROR	GET-MESSAGE-ID
USTATUS	O Rt	O Rt		O Rt						O Rt				
UWSTAT-LIFETIME										O				
UWTIME	O													
WAIT	O	O											O	

Unique Message ID

This section covers the following topics:

- Introduction
- Simple Client/Server Scenario
- Default Scenario (ACI and RPC)
- Generating a Message ID before Message is Sent

Introduction

With ACI version 11 and above, all messages sent to the broker are given a unique message ID. This message ID is useful for tracking individual messages. `PSTORE-VERSION=5` is required to save and restore message IDs of persistent data.

Under UNIX and Windows, the message ID is a Universally Unique Identifier (UUID). On mainframe platforms, the value consists of CPU ID and clock (machine instructions STCK or STCKE).

ACI functions `SEND` and `GET-MESSAGE-ID` generate message IDs. ACI version 11 must be defined. No additional programming in the application is needed.

The broker stub generates message IDs and saves the values in the field `MESSAGE-ID`. The message ID is returned immediately with function `GET-MESSAGE-ID`, or after processing of function `SEND` (without wait).

If the application receives a message from the broker - either by function `SEND` (with wait) or function `RECEIVE` (with or without wait) - the field `MESSAGE-ID` contains the message ID of the received message.

If the application receives a message from the broker by function `SEND` with wait, the message ID of the sent message is returned in field `CORRELATION-ID`.

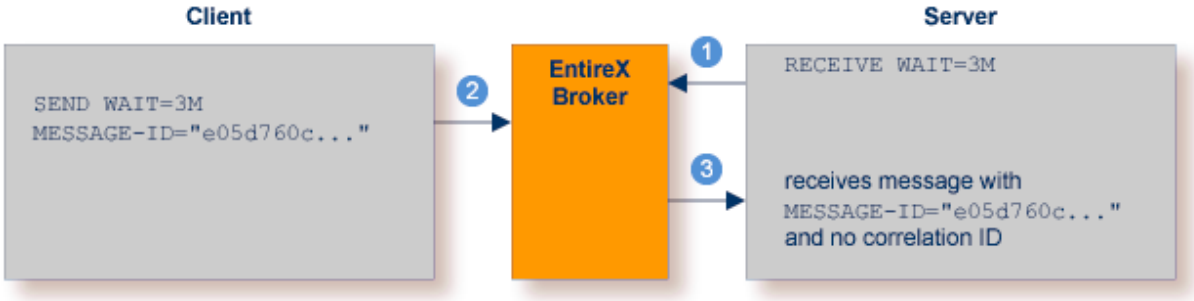
If a server receives a message from a client by function `RECEIVE` (with or without `WAIT`) and sends back a message as response to the client, we recommend moving the message ID of the received message to field `CORRELATION-ID` and to set 1 into field `USE-SPECIFIED-CORRELATION-ID` before sending the message. This correlates the response message to the received message.

If a client or a server has generated a message ID with ACI function `GET-MESSAGE-ID` and wants to use this message ID for the function `SEND` (with or waithout `WAIT`), the field `USE-SPECIFIED-MESSAGE-ID` must be set to 1.

The broker stub generates a new message ID if client or server uses function `SEND` and did not issue function `GET-MESSAGE-ID` before and set field `USE-SPECIFIED-MESSAGE-ID` to 0.

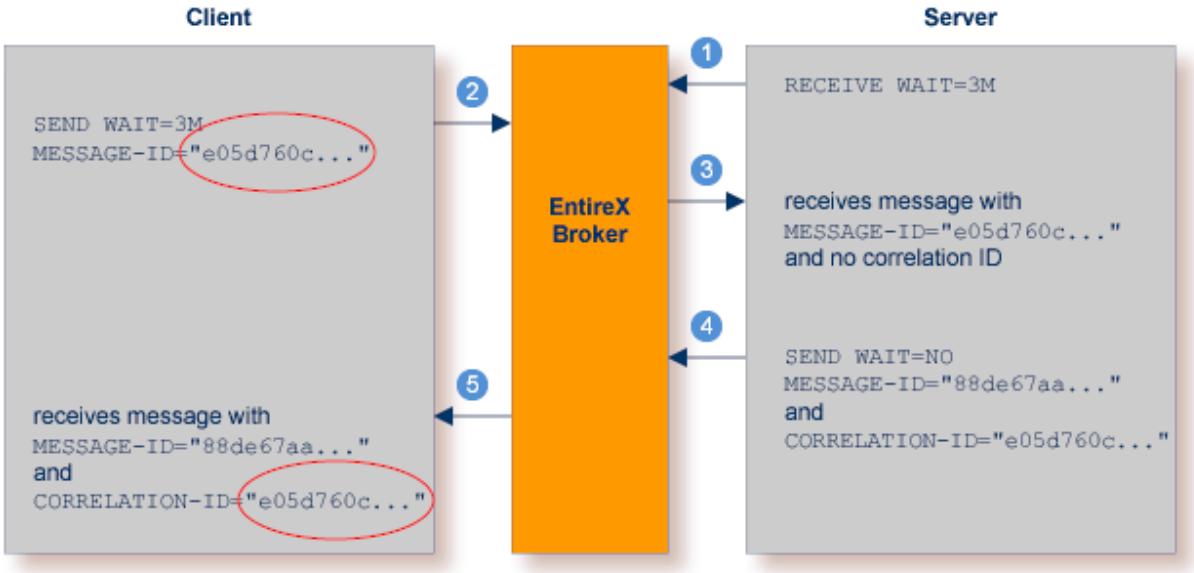
See fields `MESSAGE-ID`, `CORRELATION-ID` and `USE-SPECIFIED-MESSAGE-ID` under *Broker ACI Fields* in the ACI Programming documentation.

Simple Client/Server Scenario



1. First broker call.
2. Second broker call. Unique message ID "e05d760c..." is generated automatically. No additional programming is required.
3. Server receives message with ID generated by second broker call. In this simple scenario, no correlation ID is set.

Default Scenario (ACI and RPC)

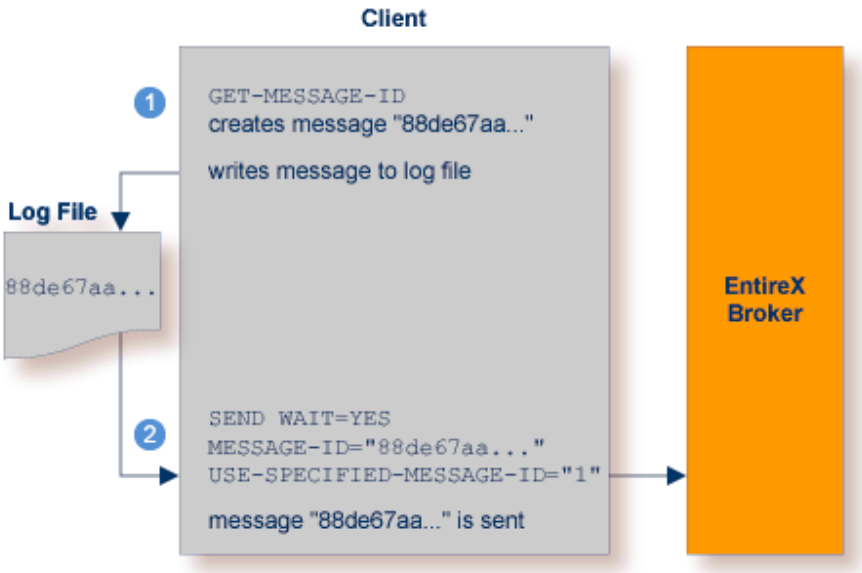


1. First broker call.
2. Second broker call. Unique message ID "e05d760c..." is generated automatically.

3. Server receives message with ID generated by second broker call. A correlation ID is not yet set. The server sets `USE-SPECIFIED-CORRELATION-ID` to 1 and moves the received message ID to the field `CORRELATION-ID`.
4. Third broker call. Another unique message ID "88de67aa..." is generated.
5. Client receives message with ID generated by third broker call. The connection to the original message is established by means of the correlation ID provided by the broker.

Generating a Message ID before Message is Sent

In this scenario a message is given a unique ID but is not sent immediately. A sample scenario would be financial transactions where the message ID needs to be logged before the actual message is sent. Function `GET-MESSAGE-ID` generates a unique message ID. The application writes the message ID to the log file. When the message is finally sent, the `SEND` request requires the flag `USE-SPECIFIED-MESSAGE-ID="1"`. This ensures no new message ID is generated; instead the unique ID created by function `GET-MESSAGE-ID` is used. This unique ID makes it easier to identify the message at a later date.



1. Function `GET-MESSAGE-ID` generates a unique message ID. The application may store this ID in a log file.
2. Broker call. A message is sent to the broker with the message ID specified earlier with the command `GET-MESSAGE-ID`.

10 Broker UOW Status Transition

- Initial UOW Status: NULL | Received 124
- Initial UOW Status: Accepted | Delivered | Postponed 125
- Initial UOW Status: Processed | Timedout 126
- Initial UOW Status: Cancelled | Discarded | Backedout 127
- Legend for UOW Status Transition Table 128
- Table of Column Abbreviations 128

This chapter contains the UOW status transition tables for EntireX Broker and covers the following topics:

See also *Broker ACI Fields* | *Broker ACI Functions* | *Error Messages and Codes*.

Initial UOW Status: NULL | Received

Initial UOW Status	Action	Resulting UOW Status				Description
		PU&PS	PU&NPS	NPU&PS	NPU&NPS	
Received	Send	Received	Received	Received	Received	
Received	Commit	Accepted	Accepted	Accepted	Accepted	
Received	ReStart	BackedOut	NULL	Discarded	NULL	
Received	BackOut	BackedOut	NULL	BackedOut	NULL	
Received	TimeOut	BackedOut	NULL	BackedOut	NULL	R6: This action can only be a conversation timeout since a UOW only exists once it is committed.
Received	Delete	Received	Received	Received	Received	
Received	Cancel	Received	Received	Received	Received	
Received	Receive	Received	Received	Received	Received	

Initial UOW Status: Accepted | Delivered | Postponed

Initial UOW Status	Action	Resulting UOW Status				Description
		PU&PS	PU&NPS	NPU&PS	NPU&NPS	
Accepted	Receive	Delivered	Delivered	Delivered	Delivered	
Accepted	Timeout	Timedout	NULL	Timedout	NULL	
Accepted	Restart	Accepted	Accepted	Discarded	NULL	
Accepted	Cancel	Cancelled	NULL	Cancelled	NULL	
Accepted	Delete	Accepted	Accepted	Accepted	Accepted	
Accepted	BackOut	Accepted	Accepted	Accepted	Accepted	
Accepted	Send	Accepted	Accepted	Accepted	Accepted	
Accepted	Commit	Accepted	Accepted	Accepted	Accepted	
Delivered	Receive	Delivered	Delivered	Delivered	Delivered	
Delivered	Commit	Processed	NULL	Processed	NULL	
Delivered	Cancel	Cancelled	NULL	Cancelled	NULL	R20: Cancel can only be issued by receiver of the UOW.
Delivered	BackOut	Accepted	Accepted	Accepted	Accepted	
Delivered	TimeOut	Timedout	NULL	NULL	NULL	
Delivered	Restart	Accepted	Accepted	Discarded	NULL	
Delivered	Delete	Delivered	Delivered	Delivered	Delivered	
Delivered	Send	Delivered	Delivered	Delivered	Delivered	
Postponed	Receive	N/A	N/A	N/A	N/A	Receive cannot be issued by any user
Postponed	Commit	N/A	N/A	N/A	N/A	Commit cannot be issued by any user.
Postponed	Cancel	Cancelled	NULL	Cancelled	NULL	Cancel can only be issued by the sender of the UOW.
Postponed	BackOut	N/A	N/A	N/A	N/A	BackOut cannot be issued by any user.
Postponed	TimeOut	Timedout	NULL	NULL	NULL	
Postponed	Restart	Accepted	Accepted	Discarded	NULL	
Postponed	Delete	N/A	N/A	N/A	N/A	Delete cannot be issued by any user.
Postponed	Send	N/A	N/A	N/A	N/A	Send cannot be issued by any user.

Initial UOW Status: Processed | Timedout

Initial UOW Status	Action	Resulting UOW Status				Description
		PU&PS	PU&NPS	NPU&PS	NPU&NPS	
Processed	Delete	NULL	N/A	NULL	N/A	Processed is a STABLE UOW status:
Processed	Timeout	NULL	NULL	NULL	N/A	All actions and transitions refer to the status of a UOW.
Processed	Restart	Processed	N/A	Processed	N/A	
Processed	Backout	Processed	N/A	Processed	N/A	
Processed	Cancel	Processed	N/A	Processed	N/A	
Processed	Commit	Processed	N/A	Processed	N/A	
Processed	Receive	Processed	N/A	Processed	N/A	
Processed	Send	Processed	N/A	Processed	N/A	
Timedout	Restart	Timeout	N/A	Timeout	N/A	Timedout is a STABLE UOW status:
Timedout	Delete	NULL	N/A	NULL	N/A	All actions and transitions refer to the status of a UOW.
Timedout	Timeout	NULL	N/A	NULL	N/A	
Timedout	Send	Timedout	N/A	Timedout	N/A	
Timedout	Receive	Timedout	N/A	Timedout	N/A	
Timedout	Commit	Timedout	N/A	Timedout	N/A	
Timedout	Backout	Timedout	N/A	Timedout	N/A	
Timedout	Cancel	Timedout	N/A	Timedout	N/A	

Initial UOW Status: Cancelled | Discarded | Backedout

Initial UOW Status	Action	Resulting UOW Status				Description
		PU&PS	PU&NPS	NPU&PS	NPU&NPS	
Cancelled	Delete	NULL	N/A	NULL	N/A	Cancelled is a STABLE UOW status:
Cancelled	Restart	Cancelled	N/A	Cancelled	N/A	All actions and transitions refer to the status of a UOW.
Cancelled	TimeOut	NULL	N/A	NULL	N/A	
Cancelled	Send	Cancelled	N/A	Cancelled	N/A	
Cancelled	Receive	Cancelled	N/A	Cancelled	N/A	
Cancelled	Commit	Cancelled	N/A	Cancelled	N/A	
Cancelled	Backout	Cancelled	N/A	Cancelled	N/A	
Cancelled	Cancel	Cancelled	N/A	Cancelled	N/A	
Discarded	Delete	N/A	N/A	NULL	N/A	Discarded is a STABLE UOW status:
Discarded	TimeOut	N/A	N/A	NULL	N/A	All actions and transitions refer to the status of a UOW.
Discarded	Restart	N/A	N/A	Discarded	N/A	
Discarded	Cancel	N/A	N/A	Discarded	N/A	
Discarded	Send	N/A	N/A	Discarded	N/A	
Discarded	Receive	N/A	N/A	Discarded	N/A	
Discarded	Commit	N/A	N/A	Discarded	N/A	
Discarded	Backout	N/A	N/A	Discarded	N/A	
BackedOut	TimeOut	NULL	N/A	NULL	N/A	BackedOut is a STABLE UOW status:
BackedOut	Cancel	BackedOut	N/A	BackedOut	N/A	All actions and transitions refer to the status of a UOW
BackedOut	Restart	BackedOut	N/A	BackedOut	N/A	
BackedOut	Send	BackedOut	N/A	BackedOut	N/A	
BackedOut	Receive	BackedOut	N/A	BackedOut	N/A	
BackedOut	Commit	BackedOut	N/A	BackedOut	N/A	
BackedOut	Delete	NULL	N/A	NULL	N/A	
BackedOut	Backout	BackedOut	N/A	BackedOut	N/A	

Legend for UOW Status Transition Table

Abbreviation	Resulting UOW Status
N/A	Not applicable
UOW Status	Error condition, message issued, no change

Table of Column Abbreviations

Abbreviation	UOW Status
PU	Persistent unit of work
PS	Persistent status
NPU	Non-persistent unit of work
NPS	Non-persistent status

11 Broker CIS Data Structures

- Command Request Structure 131
- Command Request Parameter Combinations 134
- Common Header Structure for Response Data 138
- Information Request Structure 140
- Information Reply Structures 149

EntireX Broker provides an API for Command and Information Services (CIS) that include the following: shutting down conversations, servers and services; switching trace on and off; retrieving information on clients; registering servers and services.

Command and Information Services can be accessed from any environment from which EntireX Broker can be accessed. The structures for these services are available for the programming languages Assembler, C, Natural and COBOL.

Before referring to the structure tables below, see section *Command-line Utilities*.

This chapter describes the Command and Information Services data structures..

Note: Version numbers in the tables below refer to the CIS interface version and not to the Broker version.

Command Request Structure

The request structure is given in the table below. Note possible combinations under *Command Request Parameter Combinations*.

Field Name	Format	CIS Interface Version	Comment
VERSION	I2	1	Interface version.
OBJECT-TYPE	I2	1	Specifies the object type to which the command applies: 7 BROKER 4 CONVERSATION 18 PARTICIPANT (1) 9 PSF 21 SECURITY 1 SERVER 6 SERVICE 29 TRANSPORT
COMMAND	I2	1	Valid commands: 13 ALLOW-NEWUOWMSGS 20 CLEAR-CMDLOG-FILTER 88 NO-OPERATION 17 CONNECT-PSTORE 28 DISABLE-ACCOUNTING 24 DISABLE-CMDLOG 22 DISABLE-CMDLOG-FILTER 37 DISABLE-DYN-WORKER 18 DISCONNECT-PSTORE 27 ENABLE-ACCOUNTING 23 ENABLE-CMDLOG 21 ENABLE-CMDLOG-FILTER 38 ENABLE-DYN-WORKER 14 FORBID-NEWUOWMSGS 25 PRODUCE-STATISTICS 12 PURGE 29 RESET-USER

Field Name	Format	CIS Interface Version	Comment
			31 RESUME 19 SET-CMDLOG-FILTER 42 SET-UOW-STATUS 8 SHUTDOWN 33 START 36 STATUS 32 STOP 30 SUSPEND 26 SWITCH-CMDLOG 35 TRACE-FLUSH 2 TRACE-OFF 1 TRACE-ON 34 TRAP-ERROR
OPTION	I2	1	Possible values: 3 IMMED 4 QUIESCE 11 TR_LEVEL1 12 TR_LEVEL2 13 TR_LEVEL3 14 TR_LEVEL4 15 TR_LEVEL5 16 TR_LEVEL6 17 TR_LEVEL7 18 TR_LEVEL8 20 ACCEPTED 21 CANCELLED
P-USER-ID	A28	1	Specifies the internal unique ID which is used to distinguish between several users with the same user ID. Using this field uniquely identifies a single server. The value for this field must be obtained by a previous info request. This field is used as a handle, i.e. no translation is performed.
UOWID	A16	2	Selection field. Optional. Specifies the unit of work to be purged.
UID	A32	4	Selection field. Optional. Specifies the user name for participant shutdown.

Field Name	Format	CIS Interface Version	Comment
TOKEN	A32	4	Selection field. Optional. Specifies the token name for participant shutdown.
SERVER-CLASS	A32	5	Selection field. Optional. Specifies the server class name for command log filter addition or removal.
SERVER	A32	5	Selection field. Optional. Specifies the server name for command log filter addition or removal.
SERVICE	A32	5	Selection field. Optional. Specifies the service name for command log filter addition or removal.
RESERVED	A32	5	Reserved for future use.
CONVID	A16	7	Optional. Specifies the conversation to be shut down with command SHUTDOWN.
TRANSPORTID	A3	7	Optional. Specifies the transport task. Possible values: NET|S*nn*|T*nn*. Required for commands RESUME, START, STATUS, STOP, SUSPEND.
EXCLUDE-ATTACH-SERVERS	I1	7	Optional. Exclude attach servers when shutting down a service.
SEQNO	I4	7	Optional. Specifies the sequence number of the participant (client or server) to be shut down. Can be used instead of P-USER-ID.
ERROR-NUMBER	I4	7	Specifies the error number to be used with command TRAP-ERROR.

Command Request Parameter Combinations

The following table shows all valid combinations of parameters:

Object Type	Command	Option	Comment
BROKER	CLEAR-CMDLOG-FILTER		Remove a command log filter. The command log filter can be identified using the fields UID, SERVER-CLASS, SERVER and SERVICE.
	DISABLE-ACCOUNTING		Disable accounting.
	DISABLE-CMDLOG		Disable command logging.
	DISABLE-CMDLOG-FILTER		Disable a command log filter. The command log filter can be identified using the fields UID, SERVER-CLASS, SERVER and SERVICE.
	DISABLE-DYN-WORKER		Disable the DYNAMIC-WORKER-MANAGEMENT. DYNAMIC-WORKER-MANAGEMENT=YES must be configured in the attribute file. The current number of active worker tasks will not be changed until DYNAMIC-WORKER-MANAGEMENT is enabled again.
	ENABLE-ACCOUNTING		Enable accounting.
	ENABLE-CMDLOG		Enable command logging.
	ENABLE-CMDLOG-FILTER		Enable a command log filter. The command log filter can be identified using the fields UID, SERVER-CLASS, SERVER and SERVICE.
	ENABLE-DYN-WORKER		Enable the DYNAMIC-WORKER-MANAGEMENT again. DYNAMIC-WORKER-MANAGEMENT=YES must be configured in the attribute file. DYNAMIC-WORKER-MANAGEMENT has been disabled before. Additional worker tasks can be started again, or stopped if not used.
	PRODUCE-STATISTICS		Output current statistics to the broker log.
	SET-CMDLOG-FILTER		Add a command log filter. The command log filter can be identified using the fields UID, SERVER-CLASS, SERVER and SERVICE.

Object Type	Command	Option	Comment
	SHUTDOWN		Shutdown Broker immediately.
	SWITCH-CMDLOG		Force a switch of command logging output files.
	TRACE-FLUSH		Flush all trace data kept in internal trace buffers to stderr (DD:SYSOUT). The broker-specific attribute TRMODE=WRAP is required.
	TRACE-OFF		Set trace off in Broker.
	TRACE-ON	LEVEL	Set TRACE-LEVEL on in Broker.
	TRAP-ERROR	*error number*	Modifies the setting of the broker-specific attribute TRAP-ERROR.
CONVERSATION	SHUTDOWN	*convid*	
SERVER	SHUTDOWN	IMMED	Shutdown server immediately. The server must be uniquely identified using field P_USER_ID or SEQNOand will be completely removed from the BROKER environment. The following steps will be performed: ■ Error code 00100050 will be replied to the server if it is waiting. ■ All existing conversations will be finished with EOC. ■ User will be logged off.
		QUIESCE	Shutdown server but allow existing conversations to continue. The termination is signaled to the server by error code 00100051. After this, the next call issued must be a DEREGISTER for all services (SC=*,SN=*,SV=* if more than one service is active).
PSF	ALLOW-NEWUOWMSGS		New UOW messages are allowed.
	CONNECT-PSTORE		Connect the persistent store.
	DISCONNECT-PSTORE		Disconnect the persistent store.
	FORBID-NEWUOWMSGS		New UOW messages are not allowed.
	PURGE		Remove a unit of work from the EntireX Broker persistent store (from version 2).
	SET-UOW-STATUS	ACCEPTED	Change the status of a UOW that resides in the postpone queue back to

Object Type	Command	Option	Comment
			status ACCEPTED to make it receivable again. Applies to the specified UOW only. To address all UOWs of a service, use broker command-line utility ETBCMD (z/OS | UNIX | Windows). See also *Postponing Units of Work* under *Using Persistence and Units of Work* in the Platform-independent Administration documentation.
		CANCELLED	Cancel the specified UOW that resides in the postpone queue. It performs SYNCPOINT OPTION=CANCEL of the receiver. Applies to the specified UOW only. To address all UOWs of a service, use broker command-line utility ETBCMD.
	TRACE-OFF		Set trace off in the persistent store.
	TRACE-ON	LEVEL	Set TRACE-LEVEL on in the persistent store.
PARTICIPANT	SHUTDOWN	IMMED	Shutdown participant immediately. The participant must be identified, using fields P-USER-ID, UID or TOKEN and will be completely removed from the Broker environment. The following steps will be performed: ■ Error code 00100050 will be replied to the participant, if it is waiting. ■ All existing conversations will be finished with EOC. ■ User will be logged off.
		QUIESCE	Shutdown participant but allow existing conversations to continue. The termination is signaled to the participant by error code 00100051.
		seqno	INFO requests return a *seqno* that can be used here to identify the target.
SECURITY	RESET-USER		Clear all cached security information for a user. The user must be identified using the field UID.
	TRACE-OFF		Set trace off in EntireX Security.

Object Type	Command	Option	Comment
	TRACE-ON	LEVEL	Set TRACE-LEVEL on in EntireX Security.
SERVICE	SHUTDOWN	IMMED	
		QUIESCE	
		class/server/service	
TRANSPORT	RESUME	NET | S*nn* | T*nn*	Resume NET transport or a specific SSL or TCP communicator instance.
	START	NET | S*nn* | T*nn*	Start NET transport or a specific SSL or TCP communicator instance.
	STATUS	NET | S*nn* | T*nn*	Show status of NET transport or a specific SSL or TCP communicator instance.
	STOP	NET | S*nn* | T*nn*	Stop NET transport or a specific SSL or TCP communicator instance.
	SUSPEND	NET | S*nn* | T*nn*	Suspend NET transport or a specific SSL or TCP communicator instance.
	TRACE-OFF	COM | NET | SSL | TCP	Switch trace off for all communicators (COM) or only NET, SSL or TCP communicators.
	TRACE-ON LEVEL*n*	COM | NET | SSL | TCP	Set trace level for all communicators (COM) or only NET, SSL or TCP communicators.

Common Header Structure for Response Data

This section describes the header structure (`Struct HD_CIS`), which is used by both the information services and the command service. For command-specific or information-specific structures, see *Command Request Structure* or *Information Request Structure*.

The header structure is always the first structure in the receive buffer that comes back from an information or command service request. Even receive buffers obtained with subsequent `RECEIVE` commands have this structure as the first part of the buffer. The header structure has the following layout, whereby in the Format column I = 4-byte integer value:

Field Name	Format	CIS Interface Version	Comment
`ERROR-CODE`	I4	1	Result of request. Value 0 indicates success. See *Broker Command and Information Services Error Codes*.
`TOTAL-NUM-OBJECTS`	I4	1	Total number of objects returned in object list.
`CURRENT-NUM-OBJECTS`	I4	1	Number of objects returned within current receive block.
`MAX-SC-LEN`	I4	1	Length of longest `SERVER-CLASS` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `SERVER-CLASS` field.
`MAX-SN-LEN`	I4	1	Length of longest `SERVER-NAME` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `SERVER-NAME` field.
`MAX-SV-LEN`	I4	1	Length of longest `SERVICE` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `SERVICE` field.
`MAX-UID-LEN`	I4	1	Length of longest `USER-ID` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `USER-ID` field.
`MAX-TK-LEN`	I4	1	Length of longest `TOKEN` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `TOKEN` field.
`REQUESTTIME`	I4	4	This is the time that the request was received by the Broker kernel.
`ETB-ERROR-CODE`	A8	5	This is any secondary error code from the broker kernel. See *Error Messages and Codes*.
`ETB-ERROR-TEXT`	A40	5	This is any secondary error text from the broker kernel. See *Error Messages and Codes*.
`MAX-PPC-LIB-LEN`	I4	6	Length of longest `RPC-LIB` value in total object list. This field is only relevant if the object-specific structure for the object list contains the `RPC-LIB` field.

Field Name	Format	CIS Interface Version	Comment
MAX-PPC-PGM-LEN	I4	6	Length of longest RPC-PGM value in total object list. This field is only relevant if the object-specific structure for the object list contains the RPC-PGM field.

Information Request Structure

The information services can handle many different information structures. Applications use the information request structure to specify which information structure is required. See also *Examples of Selection Criteria*.

The layout of the information request structure is shown in the following table. Fields `BLOCK-LENGTH`, `VERSION`, and `OBJECT-TYPE` are mandatory. All other fields are optional. Fields of type I or B are considered "not specified" if they contain low value. Fields of type A are considered "not specified" if they contain low value or spaces (according to the caller's character set).

Field Name	Format	CIS Interface Version	Opt/ Req	Comment
`BLOCK-LENGTH`	I4	1	R	Defines the block length of the data packages returned (without length of header.) (`RECEIVE-LENGTH` field of ACI not used in order to keep the interfaces independent.)
`VERSION`	I2	1	R	Interface version. This describes the kind and amount of information wanted and enables us to extend the information in further versions of INFO services. Valid versions are 1 and above.
`OBJECT-TYPE`	I2	1	R	Specifies the object type for which the information is required. If an object type is specified without additional selection criteria, a list of all active objects of that type is returned in accordance with the information service being addressed (`INFO` or `USER-INFO`). Possible values are: `BROKER` 7 Info on this Broker. (1) `CLIENT` 2 Info on active clients `CMDLOG_FILTER` 23 Info on command log filters `CONVERSATION` 4 Info on active conversations `NET` 24 Info on the Entire Net-Work communicator. (1) `PARTICIPANT` 18 Info on participants `POOL_USAGE` 25 Info on Broker pool usage and dynamic memory management. (1) `PSF` 9 Info on a unit of work's status `PSFADA` 12 Info on the Adabas persistent store. (1) `PSFCTREE` 20 Info on the c-tree persistent store. (1) `PSFDIV` 11 Info on the DIV persistent store. (1)

Field Name	Format	CIS Interface Version	Opt/ Req	Comment
				RESOURCE_USAGE 26 Info on Broker resource usage. [1] SECURITY 21 Info on EntireX Security. [1] SERVER 1 Info on active servers SERVICE 6 Info on active services SSL 22 Info on SSL communicators. [1] STATISTICS 27 Statistics on selected Broker resources. [1] TCP 19 Info on TCP communicators. [1] UOW_STATISTICS 31 Statistics on UOWs of selected services. USER 28 Info on all users of Broker regardless of the user type WORKER 8 Info on all workers. [1] WORKER_USAGE 30 Info on usage of worker tasks and dynamic worker management. [1] [1] No additional selection criteria are needed. Other selection criteria fields are ignored.
USER-ID	A32	1	O	Selection criteria field. This is the user ID of the client or server as specified in the field USER-ID of the EntireX Broker ACI. The value of the field is used to restrict information to related objects of a specific user.
P-USER-ID	B28	1	O	Selection criteria field. Specifies the internal unique ID which is used to distinguish between several users with the same user ID. This field uniquely identifies a client or server process. The value for this field must be obtained by a previous info request. This field is used as a handle, i.e. no translation is performed. Any value different from low value will be treated as selection value.
TOKEN	A32	1	O	Selection criteria field. Corresponds to the TOKEN field of the EntireX Broker ACI. The value restricts the information to objects of users which have specified this TOKEN value in their Broker calls.
SERVER-CLASS	A32	1	O	Selection criteria field. Corresponds to field SERVER-CLASS in the EntireX Broker ACI. The value of this field is used to restrict information to objects concerning the services registered with this class.
SERVER-NAME	A32	1	O	Selection criteria field. Corresponds to field SERVER-NAME in the EntireX Broker ACI. The value of this field is used to restrict information to objects concerning the services registered with this server name.

Field Name	Format	CIS Interface Version	Opt/ Req	Comment
SERVICE	A32	1	O	Selection criteria field. Corresponds to field SERVICE in the EntireX Broker ACI. The value of this field is used to restrict information to objects concerning the services registered with this service name.
CONV-ID	A16	1	O	Selection criteria field. Specifies the conversation ID of a conversation. Using this field uniquely identifies a conversation. The value for this field must be obtained by a previous info request.
RESERVED	I2	1		Reserved for future use.
UOWID	A16	2	O	Selection criteria field. Specifies the unit of work ID.
UOWSTATUS	I1	2	O	Selection criteria field. Specifies the unit of work status search criteria: 1 RECEIVED 2 ACCEPTED 3 DELIVERED 4 BACKEDOUT 5 PROCESSED 6 CANCELLED 7 TIMEOUT 8 DISCARDED
USERSTATUS	A32	2	O	Selection field. Specifies the user status selection value.
RECVUID	A32	2	O	Selection field. Specifies the unit-of-work receiver's user ID.
RECVTOKEN	A32	2	O	Selection field. Specifies the unit-of-work receiver's token ID.
RECVSERVER	A32	2	O	Selection field. Specifies the unit-of-work receiver's server name.
RECVSERVICE	A32	2	O	Selection field. Specifies the unit-of-work receiver's service name.
RECVCLASS	A32	2	O	Selection field. Specifies the unit-of-work receiver's class name.
CONVERSATION-TYPE	I2	5	O	Selection field. Specifies the conversation type: 1 NON-CONVERSATIONAL 2 CONVERSATIONAL

Legend

Abbreviation	Long Form / Description
Y	The field is specified.
I	The field is ignored.
N	The field is not specified; information should not be restricted by its value.

Column Abbreviation Table

Abbreviation	Long Form / Description
UID	USER-ID
RECV-UID	RECEIVER USER-ID
PUID	P-USER-ID
TK	TOKEN
RECV-TK	RECEIVER TOKEN
SC	SERVER-CLASS
RECV-SC	RECEIVER SERVER-CLASS
SN	SERVER-NAME
RECV-SN	RECEIVER SERVER-NAME
SV	SERVICE
RECV-SV	RECEIVER SERVICE
CID	CONV-ID
UOWID	UNIT OF WORK ID
UWSTAT	UNIT OF WORK STATUS
USTAT	USER STATUS

Selection Criteria CLIENT Object Type

Selection	PUID	UID	TK	SC	SN	SV	CID
Client (1)	Y	I	I	I	I	I	I
Client (2)	N	Y	Y	I	I	I	I
Clients with UID	N	Y	N	I	I	I	I
Clients with TK	N	N	Y	I	I	I	I
all clients	N	N	N	I	I	I	I

Selection Criteria SERVER Object Type

Selection	PUID	UID	TK	SC	SN	SV	CID
Server (1)	Y	I	I	I	I	I	I
Server (2)	N	Y	Y	I	I	I	I
Servers with UID	N	Y	N	I	I	I	I
Servers with TK	N	N	Y	I	I	I	I
Servers offering service	N	N	N	Y	Y	Y	I
All Servers	N	N	N	N	N	N	I

Selection Criteria SERVICE Object Type

Selection	PUID	UID	TK	SC	SN	SV	CID
Services offered by this server (1)	Y	I	I	I	I	I	I
Services offered by this server (2)	N	Y	Y	I	I	I	I
Services with this SC/SN/SV	N	N	N	Y	Y	Y	I
Services with this SC/SN	N	N	N	Y	Y	N	I
Services with this SC/SV	N	N	N	Y	N	Y	I
Services with this SC	N	N	N	Y	N	N	I
Services with this SN/SV	N	N	N	N	Y	Y	I
Services with this SN	N	N	N	N	Y	N	I
Services with this SV	N	N	N	N	N	Y	I
All services	N	N	N	N	N	N	I

Selection Criteria CONV Object Type

Selection	PUID	UID	TK	SC	SN	SV	CID
Conversations of this client/server (1)	Y	I	I	I	I	I	I
Conversations of this client/server (2)	N	Y	Y	I	I	I	I
Conversations of this service	N	N	N	Y	Y	Y	I
The conversation with CID	N	N	N	N	N	N	Y
All Conversations	N	N	N	N	N	N	N

Selection Criteria PSF Object Type (Version 2 and above)

Selection	UOWID	UID	TK	SC	SN	SV	CID
The unit of work	Y	I	I	I	I	I	I
All units of work for the conversation	N	I	I	I	I	I	Y
UOWs with client UID	N	Y	N	N	N	N	N
UOWs with clients having TK	N	N	Y	N	N	N	N
UOWs with the client SC	N	N	N	Y	N	N	N
UOWs with the client SN	N	N	N	N	Y	N	N
UOWs with the client SV	N	N	N	N	N	Y	N
UOWs with the client SC/SN	N	N	N	Y	Y	N	N
UOWs with the client SC/SV	N	N	N	Y	N	Y	N
UOWs with the client SC/SN/SV	N	N	N	Y	Y	Y	N
UOWs with the client SN/SV	N	N	N	N	Y	Y	N
All UOWs	N	N	N	N	N	N	N
UOWs with user status	I	I	I	I	I	N	Y
UOWs with UOW status	I	I	I	I	I	Y	N
UOWs with server ID	Y	N	N	N	N	N	N
UOWs with server having TK	N	N	Y	N	N	N	N
UOWs with the server SC	N	N	N	Y	N	N	N
UOWs with the server SN	N	N	N	N	Y	N	N
UOWs with the server SV	N	N	N	N	N	Y	N
UOWs with the server SC/SN	N	N	N	Y	Y	N	N
UOWs with the server SC/SV	N	N	N	Y	N	Y	N
UOWs with the server SC/SN/SV	N	N	N	Y	Y	Y	N
UOWs with the server SN/SV	N	N	N	N	Y	Y	N
All UOWs	N	N	N	N	N	N	N

Key

(1) if participant is *not* using TK (token) for authentication

(2) if participant is using TK (token) for authentication

Information Reply Structures

The information reply structures are defined and described in the delivered source code. The structures are available for programming languages Assembler, C, Natural and COBOL.

- BROKER-OBJECT (Struct INFO_BKR)
- CLIENT-SERVER-PARTICIPANT-OBJECT (Struct INFO_CS)
- CMDLOG_FILTER-OBJECT (Struct INFO_CMDLOG_FILTER)
- CONVERSATION-OBJECT (Struct INFO_CV)
- NET-OBJECT (Struct INFO_NET)
- POOL-USAGE-OBJECT (Struct INFO_POOL_USAGE)
- PSF-OBJECT (Struct INFO_PSF)
- PSFADA-OBJECT (Struct INFO_PSFADA)
- PSFCTREE-OBJECT (Struct INFO_PSTCTREE)
- PSFDIV-OBJECT (Struct INFO_PSFDIV)
- RESOURCE-USAGE-OBJECT (Struct INFO_RESOURCE_USAGE)
- SECURITY-OBJECT (Struct INFO_SECURITY)
- SERVICE-OBJECT (Struct INFO_SV)
- SSL-OBJECT (Struct INFO_SSL)
- STATISTICS-OBJECT (Struct INFO_STATISTICS) (Excerpt of BROKER-OBJECT)
- TCP-OBJECT (Struct INFO_TCP)
- UOW-STATISTICS (Struct INFO_UOW_STATISTICS)
- USER-OBJECT (Struct INFO_USER)
- WORKER-OBJECT (Struct INFO_WKR)
- WORKER-USAGE-OBJECT (Struct INFO_WORKER-USAGE)

BROKER-OBJECT (Struct INFO_BKR)

Field Name	Format	CIS Interface Version	Description / Action
PLATFORM	A8	1	Platform dependent.
RUNTIME	I4	1	Time since Broker started, in seconds. Computed from current time - Broker start time.
NUM-WORKER-ACT	I4	1	Number of active workers.
NUM-LONG	I4	1	Number of long buffers defined (see NUM-LONG-BUFFER).
LONG-ACT	I4	1	Number of long buffers active (in use).
LONG-HIGH	I4	1	Highest number of long buffers active since Broker started.
NUM-SHORT	I4	1	Number of short buffers defined (see NUM-SHORT-BUFFER).
SHORT-ACT	I4	1	Number of short buffers active.
SHORT-HIGH	I4	1	Highest number of short buffers active since Broker started.
LONG-SIZE	I4	1	Size of long buffer entry.
SHORT-SIZE	I4	1	Size of short buffer entry.
NUM-SERVICE	I4	1	Number of services defined (see NUM-SERVICE).
SERVICE-ACT	I4	1	Number of services active.
NUM-SERVER	I4	1	Number of servers defined (see NUM-SERVER).
SERVER-ACT	I4	1	Number of servers active. This counter also includes the active Attach Server instances.
SERVER-HIGH	I4	1	Highest number of servers active since Broker started.
NUM-CLIENT	I4	1	Number of clients defined (see NUM-CLIENT).
CLIENT-ACT	I4	1	Number of clients active.
CLIENT-HIGH	I4	1	Highest number of clients active since Broker started.
NUM-CONV	I4	1	Number of conversations defined (see NUM-CONVERSATION).
CONV-HIGH	I4	1	Highest number of conversations active since Broker started.
TRACE-LEVEL	I2	1	Actual Trace Level value.
UNUSED1	I2	1	Unused.
LMAXUOWS	I4	2	Maximum number of active UOWs.
LMAXUOWMSG	I4	2	Maximum number of messages in a UOW.

Field Name	Format	CIS Interface Version	Description / Action
LUWTIME	I4	2	Maximum UOW lifetime
LMAXDELCNT	I4	2	Currently not in use. (Count is always zero.)
LMAXMSGSIZE	I4	2	Maximum size of a message
LTOTALUOWS	I4	2	Number of UOWs.
CSTORE	I1	2	Store attribute for all UOWs: 0 OFF 1 BROKER
CPSTORE	I1	2	Startup value for persistent store: 0 NO 1 HOT 2 COLD 4 WARM
CUWSTATP	I1	2	UOW status lifetime multiplier (0-255)
CDEFERRED	I1	2	Default status attribute for all UOWs: 0 NO 1 YES
CACCOUNTING	A3	3	NO Accounting not active YES Accounting active on UNIX and Windows *nnn* SMF Record number on z/OS
CAUTHDEFAULT	I1	3	Authorization Default: 0 NO 1 YES
LSSLPORT	I4	3	Port number being used for SSL transport (UNIX and Windows only).
NEW-UOW-MESSAGES	I1	3	New UOW messages: 0 NO 1 YES
UNUSED2	I1	3	Unused.
CPLATNAME	A32	3	Full platform name where Broker is running

Field Name	Format	CIS Interface Version	Description / Action
CPSTORETYPE	A8	3	Persistent store type. It will be one of the following values: DIV Data-in-Virtual Persistent Store (z/OS only) FILE B-Tree Store (UNIX and Windows only, no longer supported) ADABAS Adabas Persistent Store (all platforms)
HIGHEST-API-VERSION	I1	4	For example: 0x06.
HIGHEST-CIS-VERSION	I1	4	For example: 0x06.
PSTORE-CONNECTED	I1	4	0 NO 1 YES
ATTACH-MGRS-ACT	I4	4	Number of attach servers active.
LUWSTAT-ADD-TIME	I4	4	Unit of work status additional lifetime.
PRODUCT-VERSION	A16	4	Version, release, service pack, and patch level, e.g. 8.0.1.00.
LICENSE-EXPIRATION-DATE	A10	5	License expiration date.
SECURITY-TYPE	I1	5	Security type: 0 None 1 SAG 2 Light 3 Other
ACCOUNTING-ENABLED	I1	5	1 Accounting enabled 0 Accounting disabled
NUM-FREE-CCB	I4	5	Number of free CCB entries (conversation control block).
NUM-FREE-PCB	I4	5	Number of free PCB entries(participant control block).
NUM-FREE-PCBEXT	I4	5	Number of free PCBEXT entries (PCB extension).
NUM-FREE-SCB	I4	5	Number of free SCB entries (service control block).
NUM-FREE-SCBEXT	I4	5	Number of free SCBEXT entries (SCB extension).
NUM-FREE-TCBEXT	I4	5	Number of free TCBEXT entries (TCP extension).
NUM-FREE-TOQ	I4	5	Number of free TOQ entries (timeout queue).
NUM-FREE-UWCB	I4	5	Number of free UWCB entries (UOW control block).

Field Name	Format	CIS Interface Version	Description / Action
NUM-COM-BUFFER	I4	5	Number of communication buffers.
NUM-COM-SLOT	I4	5	Number of communication buffer slots.
NUM-COM-SLOT-FREE	I4	5	Number of communication buffer slots free.
NUM-CMDLOG-FILTER	I4	5	Number of CMDLOG filters.
NUM-CMDLOG-FILTER-ACTIVE	I4	5	Number of CMDLOG filters active.
CMDLOG	I1	5	Reflects status of Broker attribute CMDLOG: 1 Command logging features are available for the Broker 0 Command logging not available
CMDLOG-ENABLED	I1	5	Reflects result of commands DISABLE-CMDLOG and ENABLE-CMDLOG: 1 Command logging enabled 0 Command logging temporarily disabled
NOTUSED3	A2	5	Alignment.
ATTRIBUTE-FILE-NAME	A256	5	Attribute file name.
LOG-FILE-NAME	A256	5	Name of trace log file.
LOG-FILE-SIZE	I4	5	Size of trace log file.
LICENSE-FILE-NAME	A256	5	License file name.
CMDLOG-FILE-SIZE	I4	5	Max. size of CMDLOG file.
OPEN-CMDLOG-FILE-NAME	A256	5	Name of open CMDLOG file.
OPEN-CMDLOG-FILE-SIZE	I4	5	Size of CMDLOG file.
CLOSED-CMDLOG-FILE-NAME	A256	5	Name of closed CMDLOG file.
CLOSED-CMDLOG-FILE-SIZE	I4	5	Size of closed CMDLOG file.
RESERVED	I4	5	Reserved for future use.
ACCOUNTING-FILE-NAME	A256	5	Name of accounting output file.
ACCOUNTING-FILE-SIZE	I4	5	Size of accounting output file.
CONTROL-INTERVAL	I4	5	Control interval in seconds.
MAX-TAKEOVER-ATTEMPTS	I4	5	Max. number of takeover attempts.
RUN-MODE	A16	5	Broker run mode.
PARTNER-CLUSTER-ADDRESS	A32	5	Partner Cluster Address.
CMDLOG-SWITCHES-BY-SIZE	I4	5	Number of CMDLOG switches by size.
CMDLOG-SWITCHES-BY-CIS	I4	5	Number of CMDLOG switches by CIS.
CLIENT-NONACT	I4	7	Client timeout in seconds. See broker attribute CLIENT-NONACT.

Field Name	Format	CIS Interface Version	Description / Action
NUM-WQE	I4	7	Number of work queue entries. See broker attribute NUM-WQE.
TOTAL-STORAGE-ALLOCATED	I4	7	Size of allocated storage in bytes.
TOTAL-STORAGE-ALLOCATED-HIGH	I4	7	Highest size of allocated storage in bytes since Broker started.
TOTAL-STORAGE-LIMIT	I4	7	Maximum of storage that can be allocated. See broker attribute MAX-MEMORY.
BROKER-ID	A32	7	BROKER-ID. See broker attribute BROKER-ID.
HOST-NAME	A256	7	Name of host running broker (on z/OS copied from CVTSNAME).
SYSPLEX-NAME	A8	7	Name of SYSPLEX (copied from ECVTSPLX).
CAUTOLOGON	I1	7	Auto logon: 0 NO 1 YES See broker attribute AUTOLOGON.
CDYNAMIC-MEMORY-MANAGEMENT	I1	7	Dynamic memory management: 0 NO 1 YES See broker attribute DYNAMIC-MEMORY-MANAGEMENT.
CDYNAMIC-WORKER-MANAGEMENT	I1	7	Dynamic worker management: 0 NO 1 YES See broker attribute DYNAMIC-WORKER-MANAGEMENT.
CSERVICE-UPDATES	I1	7	Service updates: 0 NO 1 YES See broker attribute SERVICE-UPDATES.
CTRANSPORT-NET	I1	7	Was TRANSPORT=NET specified? 0 NO 1 YES See broker attribute TRANSPORT=NET.

Field Name	Format	CIS Interface Version	Description / Action
CTRANSPORT-SSL	I1	7	Was TRANSPORT=SSL specified? 0 NO 1 YES See broker attribute TRANSPORT=SSL.
CTRANSPORT-TCP	I1	7	Was TRANSPORT=TCP specified? 0 NO 1 YES See broker attribute TRANSPORT=TCP.
NTRAP-ERROR	I4	7	Value defined for attribute TRAP-ERROR.
CPU-USED-IN-SECONDS	I4	9	Amount of CPU time in seconds used by Broker process since Broker start.
CPU-USED-REST-IN-MICROSECONDS	I4	9	Additional CPU time in microseconds used by Broker process since Broker start. (CPU time is provided by two fields because total value including microseconds may exceed one 4-byte integer.)
CPU-USED-PERCENTAGE	I4	9	CPU time consumed by Broker process in relation to total CPU workload in percent and normalized by the number of CPUs. It never exceeds 100%.

CLIENT-SERVER-PARTICIPANT-OBJECT (Struct INFO_CS)

Field Name	Format	CIS Interface Version	Description / Action
USER-ID	A32	1	Corresponds to USER-ID in the ACI. The maximum length of this field is determined by field MAX-UID-LEN in the header. See *Common Header Structure for Response Data*.
P-USER-ID	B28	1	Specifies the physical internal unique ID which is used to distinguish between several users with the same user ID. This field is used as a handle, i.e. no translation is performed. With CIS commands SHUTDOWN PARTICIPANT and SHUTDOWN SERVER, field SEQNO is provided as unique criterion.
P-USER-ID-CHAR	A28	1	No longer used.
TOKEN	A32	1	Corresponds to TOKEN in the ACI. The maximum length of this field is determined by MAX-TK-LEN in the header. See *Common Header Structure for Response Data*.
CHAR-SET	I2	1	Character set of user's platform: 34 EBCDIC IBM 66 EBCDIC SNI 1 ASCII PC 386 16 ASCII PC OS/2 128 ASCII 8859-1
ENDIAN	I2	1	Endian type of user's platform: 1 Big endian (high order first) 0 Little endian
STATUS	I2	1	Status of user: 0 Not waiting 5 Waiting
UNUSED1	I2	1	Unused.
WAIT-CONV-TYPE	A16	1	Only valid if user is waiting. Indicates what kind of conversation user is waiting for: NEW User waiting for new conversations ANY User waiting for any conversation OLD User waiting for old conversations

Field Name	Format	CIS Interface Version	Description / Action
			NONE User waiting for non-conversational reply CONV-ID User waiting for specific conversation
WAIT-SERVER-CLASS	A32	1	When waiting for ANY, NEW or OLD, the class name of the service to wait for is returned.
WAIT-SERVER-NAME	A32	1	When waiting for ANY, NEW or OLD, the server name of the service to wait for is returned.
WAIT-SERVICE	A32	1	When waiting for ANY, NEW or OLD, the name of the service to wait for is returned.
CONV-ACT	I4	1	Number of active conversations of this user.
SERVICE-ACT	I4	1	Number of services active (offered) by this server. This information is available for server only.
LAST-ACTIVE	I4	1	Elapsed time since the last activity of the user.
NONACT	I4	1	Non-activity time-out value.
WAIT-NEW	I4	1	Accumulated time a server waited for new conversations. (Receive with CONVID=NEW or CONVID=ANY). A high value indicates that server has capacity.
NUM-WAIT-NEW	I4	1	Number of times a server had to wait for new conversations.
WAIT-OLD	I4	1	Accumulated time a server or client waited for messages of existing conversations. (Receive with CONVID=*cid* or CONVID=OLD.) A high value for a server indicates that server had to wait for the clients. A high value for a client indicates that the server's response was delayed.
NUM-WAIT-OLD	I4	1	Number of times a server or client had to wait for messages of existing conversations.
SUM-CONV	I4	1	Sum of conversations (including non-conversational requests) for the user since start of User.
LTOTALUOWS	I4	2	Number of UOWs.
IP-ADDRESS	A16	4	IPv4 address of client/server.
HOST-NAME	A256	4	Host name of client/server.
RECV-OPTION	I1	4	Receive option.
ATTACH-MGR	I1	4	Attach manager indicator.
UNUSED2	I2	4	Unused.
RESERVED_ETBINFO_V73_1	A32	5	Reserved for future use.

Field Name	Format	CIS Interface Version	Description / Action
APPLICATION-NAME	A64	5	The name of the executable that called the broker. If the program that issued the broker call is running on a mainframe system, the eight-byte job name is used as application name. If the job name is shorter than eight bytes, it is padded with underscore characters. If the z/OS program issuing the broker call is running in a TP monitor (except IDMS/DC), a dash sign is set as ninth byte. The following eight bytes from position 10-17 contain monitor-dependent data: CICS The four-byte transaction ID is set. Com-plete The eight-byte program name is set. IMS The four-byte IMS ID is set. Padding blanks in bytes 10-17 are replaced by underscore characters.
APPLICATION-TYPE	A8	5	Application type. This field is used internally. It can be set by other Software AG products, which pass this value to the Broker stub via an unpublished control block. If no value is set, the respective operating system is displayed here.
RESERVED_ETBINFO_V73_3	A32	5	Reserved for future use.
COUNT-AUTHORIZATION-SUCCEEDED	I4	5	Counter AUTHORIZ succeeded.
COUNT-AUTHORIZATION-FAILED	I4	5	Counter AUTHORIZ failed.
CREATE-TIME	I4	5	Creation time of the participant. Time as time_t value (number of seconds since the epoch (00:00:00 UTC, January 1, 1970)).
RPC-LIBRARY-NAME	A128	6	Name of the RPC library of the current user request. If the user is inactive at the time of the request and has not issued a request to be processed by the Broker, no RPC information is displayed.
RPC-PROGRAM-NAME	A128	6	Name of the RPC program of the current user request. If the user is inactive at the time of the request and has not issued a request to be processed by the Broker, no RPC information is displayed.
SEQNO	I4	7	Unique sequence number of client/server. Can be used with CIS command SHUTDOWN.
APPLICATION-VERSION	A16	7	Application version. This field is used internally. It can be set by other Software AG products, which pass this value to the Broker stub via an unpublished

Field Name	Format	CIS Interface Version	Description / Action
			control block. The value is the version of the program that calls the Broker stub.
IPV6-ADDRESS	A46	8	IPv6 address corresponding to attribute HOST in DEFAULTS=SSL|TCP section of Broker attribute file.
NOTUSED8	A2	8	Alignment.
ARF	I1	10	0 Ordinary client or server. 1 Client or server is an Adabas Event Replicator source or target.
SCM	I1	10	If ARF=1: 0 Single Conversation Mode was not activated by the client. 1 Single Conversation Mode was activated by the client.
PREFETCH	I1	10	If ARF=1 and SCM=1: 0 Server is not prefetching units of work 1 Server is prefetching units of work
ROAMING	I1	10	0 Server is not using roaming of conversations. 1 Server is using roaming of conversations.
RPC	A5	10	YES Server identified as RPC server. NO Server identified as ACI server. UNDEF Server not identified at all.
NOTUSED10	A3	10	Alignment.

CMDLOG_FILTER-OBJECT (Struct INFO_CMDLOG_FILTER)

Field Name	Format	CIS Interface Version	Description / Action
UID	A32	5	User ID.
SERVER-CLASS	A32	5	Class.
SERVER	A32	5	Server.
SERVICE	A32	5	Service.

Field Name	Format	CIS Interface Version	Description / Action
SETTER-UID	A32	5	User ID of filter setter.
ENABLED	I1	5	1 Enabled 0 Disabled

CONVERSATION-OBJECT (Struct INFO_CV)

Field Name	Format	CIS Interface Version	Description / Action
CONV-ID	A16	1	Unique identification of conversation.
SERVER-USER-ID	A32	1	User ID of server - corresponds to USER-ID in the ACI. The maximum length of this field is determined by field MAX-UID-LEN in the header. See *Common Header Structure for Response Data*.
SERVER-P-USER-ID	B28	1	Specifies the physical internal unique ID which is used to distinguish between several users with the same user ID. This field is used as a handle i.e. no translation is performed.
SERVER-P-USER-ID-CHAR	A28	1	No longer used.
SERVER-TOKEN	A32	1	Partner's additional identification - corresponds to TOKEN in the ACI. The maximum length of this field is determined by MAX-TK-LEN in the header. See *Common Header Structure for Response Data*.
CLIENT-USER-ID	A32	1	Owners name. Corresponds to USER-ID in the ACI.
CLIENT-P-USER-ID	B28	1	Specifies the physical internal unique ID which is used to distinguish between several users with the same user ID. This field is used as a handle i.e. no translation is performed.
CLIENT-P-USER-ID-CHAR	A28	1	No longer used.
CLIENT-TOKEN	A32	1	Owner's additional identification - corresponds to TOKEN in the ACI.
SERVER-CLASS	A32	1	Server class of Service of Conversation.
SERVER-NAME	A32	1	Server name of Service of Conversation. The maximum length of SERVER-CLASS, SERVER-NAME and SERVICE is determined by fields MAX-SC-LEN, MAX-SN-LEN and MAX-SV-LEN in the header. See *Common Header Structure for Response Data*.
SERVICE	A32	1	Service name of Service of Conversation.
CONV-TIME-OUT	I4	1	Conversation timeout (corresponds to CONV-NONACT of the service in the attribute file)

Field Name	Format	CIS Interface Version	Description / Action
LAST-ACTIVE	I4	1	Elapsed time since the last activity for this conversation.
TYPE	I2	1	Type of conversation: 0 conversational 1 non-conversational
UNUSED1	I2	2	Unused.
LTOTALUOWS	I4	2	Number of UOWs.
CLIENT-RPC-LIBRARY-NAME	A128	6	Name of the RPC library that was provided by the RPC client at the start of the conversation, that is, the first SEND that contains both RPC library and RPC program is stored in the conversation.
CLIENT-RPC-PROGRAM-NAME	A128	6	Name of the RPC program that was provided by the RPC client at the start of the conversation, that is, the first SEND that contains both RPC library and RPC program is stored in the conversation.
SERVER-RPC-LIBRARY-NAME	A128	6	Name of the RPC library that was provided by the RPC server with the first response to clients request, that is, the first SEND that contains both RPC library and RPC program is stored in the conversation.
SERVER-RPC-PROGRAM-NAME	A128	6	Name of the RPC program that was provided by the RPC server with the first response to clients request, that is, the first SEND that contains both RPC library and RPC program is stored in the conversation.
ARF	I1	10	0 Ordinary conversation. 1 Conversation belongs to Adabas Event Replication.
SCM	I1	10	If ARF=1: 0 Conversation does not run in Single Conversation Mode. 1 conversation runs in Single Conversation Mode.
PREFETCH	I1	10	If ARF=1 and SCM=1: 0 Server is not prefetching units of work. 1 Server is prefetching units of work.
ROAMING	I1	10	0 Server is not using roaming of conversations. 1 Server is using roaming of conversations.

NET-OBJECT (Struct INFO_NET)

Field Name	Format	CIS Interface Version	Description / Action
CLONE-INDEX	I4	5	Clone index.
STATUS	I4	5	Status of communicator. Possible values defined as ETB_INFO_COM_STATUS_.
DBID	I2	5	DBID.
SVC-NUMBER	I2	5	Adabas SVC number.
IUBL	I4	5	Maximum buffer length.
TIME	I4	5	MPM-12 timeout.
NABS	I4	5	Number of attached buffers.
CQES	I4	5	Number of CQEs.
FORCE	I1	5	DBID table entry overwrite.
LOCAL	I1	5	1 Local node 0 Not local
NOTUSED0	A2	5	Alignment.

POOL-USAGE-OBJECT (Struct INFO_POOL_USAGE)

Field Name	Format	CIS Interface Version	Description / Action
TOTAL-NUM-POOLS	I4	7	Number of pools currently allocated.
TOTAL-STORAGE-ALLOCATED	I4	7	Size of allocated storage in bytes.
ACCOUNTING-NUM-POOLS	I4	7	ACCOUNTING: Number of pools.
ACCOUNTING-SIZE-ALL-POOLS	I4	7	ACCOUNTING: Size of all pools in bytes.
ACCOUNTING-SIZE-ONE-POOL	I4	7	ACCOUNTING: Size of one pool in bytes.
BLACKLIST-NUM-POOLS	I4	7	BLACKLIST: Number of pools.
BLACKLIST-SIZE-ALL-POOLS	I4	7	BLACKLIST: Size of all pools in bytes.
BLACKLIST-SIZE-ONE-POOL	I4	7	BLACKLIST: Size of one pool in bytes.
BROKER-TO-BROKER-NUM-POOLS	I4	7	BROKER-TO-BROKER: Number of pools.
BROKER-TO-BROKER-SIZE-ALL-POOLS	I4	7	BROKER-TO-BROKER: Size of all pools in bytes.
BROKER-TO-BROKER-SIZE-ONE-POOL	I4	7	BROKER-TO-BROKER: Size of one pool in bytes.
COM-BUFFER-NUM-POOLS	I4	7	COM-BUFFER: Number of pools.
COM-BUFFER-SIZE-ALL-POOLS	I4	7	COM-BUFFER: Size of all pools in bytes.
COM-BUFFER-SIZE-ONE-POOL	I4	7	COM-BUFFER: Size of one pool in bytes.

Field Name	Format	CIS Interface Version	Description / Action
CMDLOG-NUM-POOLS	I4	7	CMDLOG: Number of pools.
CMDLOG-SIZE-ALL-POOLS	I4	7	CMDLOG: Size of all pools in bytes.
CMDLOG-SIZE-ONE-POOL	I4	7	CMDLOG: Size of one pool in bytes.
CONNECTION-NUM-POOLS	I4	7	CONNECTION: Number of pools.
CONNECTION-SIZE-ALL-POOLS	I4	7	CONNECTION: Size of all pools in bytes.
CONNECTION-SIZE-ONE-POOL	I4	7	CONNECTION: Size of one pool in bytes.
CONVERSATION-NUM-POOLS	I4	7	CONVERSATION: Number of pools.
CONVERSATION-SIZE-ALL-POOLS	I4	7	CONVERSATION: Size of all pools in bytes.
CONVERSATION-SIZE-ONE-POOL	I4	7	CONVERSATION: Size of one pool in bytes.
HEAP-NUM-POOLS	I4	7	HEAP: Number of pools.
HEAP-SIZE-ALL-POOLS	I4	7	HEAP: Size of all pools in bytes.
HEAP-SIZE-ONE-POOL	I4	7	HEAP: Size of one pool in bytes.
MSG-BUFFER-LONG-NUM-POOLS	I4	7	MSG-BUFFER-LONG: Number of pools.
MSG-BUFFER-LONG-SIZE-ALL-POOLS	I4	7	MSG-BUFFER-LONG: Size of all pools in bytes.
MSG-BUFFER-LONG-SIZE-ONE-POOL	I4	7	MSG-BUFFER-LONG: Size of one pool in bytes.
MSG-BUFFER-SHORT-NUM-POOLS	I4	7	MSG-BUFFER-SHORT: Number of pools.
MSG-BUFFER-SHORT-SIZE-ALL-POOLS	I4	7	MSG-BUFFER-SHORT: Size of all pools in bytes.
MSG-BUFFER-SHORT-SIZE-ONE-POOL	I4	7	MSG-BUFFER-SHORT: Size of one pool in bytes.
PARTICIPANT-NUM-POOLS	I4	7	PARTICIPANT: Number of pools.
PARTICIPANT-SIZE-ALL-POOLS	I4	7	PARTICIPANT: Size of all pools in bytes.
PARTICIPANT-SIZE-ONE-POOL	I4	7	PARTICIPANT: Size of one pool in bytes.
PARTICIPANT-EXT-NUM-POOLS	I4	7	PARTICIPANT-EXT: Number of pools.
PARTICIPANT-EXT-SIZE-ALL-POOLS	I4	7	PARTICIPANT-EXT: Size of all pools in bytes.
PARTICIPANT-EXT-SIZE-ONE-POOL	I4	7	PARTICIPANT-EXT: Size of one pool in bytes.
PROXY-QUEUE-NUM-POOLS	I4	7	PROXY-QUEUE: Number of pools.
PROXY-QUEUE-SIZE-ALL-POOLS	I4	7	PROXY-QUEUE: Size of all pools in bytes.
PROXY-QUEUE-SIZE-ONE-POOL	I4	7	PROXY-QUEUE: Size of one pool in bytes.
SERVICE-ATTRIBUTES-NUM-POOLS	I4	7	SERVICE-ATTRIBUTES: Number of pools.
SERVICE-ATTRIBUTES-SIZE-ALL-POOLS	I4	7	SERVICE-ATTRIBUTES: Size of all pools in bytes.
SERVICE-ATTRIBUTES-SIZE-ONE-POOL	I4	7	SERVICE-ATTRIBUTES: Size of one pool in bytes.
SERVICE-NUM-POOLS	I4	7	SERVICE: Number of pools.
SERVICE-SIZE-ALL-POOLS	I4	7	SERVICE: Size of all pools in bytes.
SERVICE-SIZE-ONE-POOL	I4	7	SERVICE: Size of one pool in bytes.

Field Name	Format	CIS Interface Version	Description / Action
SERVICE-EXT-NUM-POOLS	I4	7	SERVICE-EXT: Number of pools.
SERVICE-EXT-SIZE-ALL-POOLS	I4	7	SERVICE-EXT: Size of all pools in bytes.
SERVICE-EXT-SIZE-ONE-POOL	I4	7	SERVICE-EXT: Size of one pool in bytes.
TIMEOUT-QUEUE-NUM-POOLS	I4	7	TIMEOUT-QUEUE: Number of pools.
TIMEOUT-QUEUE-SIZE-ALL-POOLS	I4	7	TIMEOUT-QUEUE: Size of all pools in bytes.
TIMEOUT-QUEUE-SIZE-ONE-POOL	I4	7	TIMEOUT-QUEUE: Size of one pool in bytes.
TRANSLATION-NUM-POOLS	I4	7	TRANSLATION: Number of pools.
TRANSLATION-SIZE-ALL-POOLS	I4	7	TRANSLATION: Size of all pools in bytes.
TRANSLATION-SIZE-ONE-POOL	I4	7	TRANSLATION: Size of one pool in bytes.
UOW-NUM-POOLS	I4	7	UOW: Number of pools.
UOW-SIZE-ALL-POOLS	I4	7	UOW: Size of all pools in bytes.
UOW-SIZE-ONE-POOL	I4	7	UOW: Size of one pool in bytes.
WORK-QUEUE-NUM-POOLS	I4	7	WORK-QUEUE: Number of pools.
WORK-QUEUE-SIZE-ALL-POOLS	I4	7	WORK-QUEUE: Size of all pools in bytes.
WORK-QUEUE-SIZE-ONE-POOL	I4	7	WORK-QUEUE: Size of one pool in bytes.

PSF-OBJECT (Struct INFO_PSF)

Information about individual UOWs, or groups of UOWs, can be obtained through information services.

Field Name	Format	CIS Interface Version	Description / Action
UOWID	A16	2	Unit of work ID.
CONVID	A16	2	Conversation ID.
SENDERUID	A32	2	Sender user ID.
SENDERTOKEN	A32	2	Sender user token
SENDERSERVER	A32	2	Sender server name
SENDERCLASS	A32	2	Sender server class
SENDERSERVICE	A32	2	Sender service name
RECVRUID	A32	2	Receiver user ID.
RECVRTOKEN	A32	2	Receiver user token
RECVRSERVER	A32	2	Receiver server name
RECVRCLASS	A32	2	Receiver server class
RECVRSERVICE	A32	2	Receiver service name

Field Name	Format	CIS Interface Version	Description / Action
USERSTATUS	A32	2	User status
UWSTATUS	I1	2	UOW status: 1 RECEIVED 2 ACCEPTED 3 DELIVERED 4 BACKEDOUT 5 PROCESSED 6 CANCELLED 7 TIMEOUT 8 DISCARDED
CEOC	I1	2	End of conversation state: 0 NO 1 YES
CSTORE	I1	2	Persistence flag: 0 none 1 off 2 Broker
CUOWSTATSTORE	I1	2	Multiplier used to calculate lifetime for the persistent status of a UOW: 255 no persistent status 1-254 valid multiplier values
LEOCREASON	I4	2	End of conversation reason code.
LATTEMPTCOUNT	I4	2	Attempted delivery count.
LMSQCNT	I4	2	Number of messages.
LMSQSIZE	I4	2	Total message size.
UWSTATUSLIFETIME	A32	2	Status lifetime.
UWCREATETIME	A32	2	Pseudo time UOW created. Broker downtimes are subtracted.
UWLIFETIME	I4	2	UOW lifetime.
ARF	I1	10	0 Ordinary unit of work. 1 Unit of work belongs to Adabas Event Replication.

Field Name	Format	CIS Interface Version	Description / Action
SCM	I1	10	If ARF=1: 0 Unit of work was not created in Single Conversation Mode. 1 Unit of work was created in Single Conversation Mode.
MAX-POSTPONE-ATTEMPTS	I4	10	Number of POSTPONE-ATTEMPTS configured for the related service.
REMAINING-POSTPONE-ATTEMPTS	I4	10	Remaining attempts moving the UOW to the postpone queue.
POSTPONE-DELAY	I4	10	POSTPONE-DELAY in number of seconds configured for the related service.
TIME-BACK-TO-ACCEPTED	A32	10	Time when UOW will be taken out of the postpone queue for another processing attempt.
COMMIT-TIME	A32	10	Time UOW was committed.
CREATE-TIME	A32	10	Time UOW was created.

PSFADA-OBJECT (Struct INFO_PSFADA)

Note: Some of the fields listed in this table are represented by blanks or zeros under Windows. Such fields will not be displayed under Windows because of this limitation.

Field Name	Format	CIS Interface Version	Description / Action
ADA-INFO-VERS	I4	3	Adabas persistent store information services version.
ADA-DBID	I4	3	Adabas database ID (DBID) where the store is located.
ADA-FNR	I4	3	Adabas file number of the store (FNR).
ADA-FNAME	A16	3	Adabas file name of the store.
ADA-FORMAT-TOD	A16	3	TOD of persistent store last format in YYYMMDDHHMMSST.
ADA-FORMAT-VERS	I4	3	Persistent store format version.
ADA-START-CNT	I4	3	Number of times the persistent store has been opened.
ADA-START-TOD	A16	3	TOD of persistent store last open in YYYMMDDHHMMSST.
ADA-ATTLEN	I4	3	Length of attribute data.
ADA-OID-LEN	I4	3	Length of object identifier.
ADA-OID-OFF	I4	3	Offset of object identifier.
ADA-ATT-CNT	I4	3	Number of attributes in the store.
ADA-OID-CNT	I4	3	Number of object identifiers in the store.

Field Name	Format	CIS Interface Version	Description / Action
ADA-UI-EXTS	I4	3	Number of upper index extents of the Adabas file used by the store.
ADA-NI-EXTS	I4	3	Number of normal index extents of the Adabas file used by the store.
ADA-AC-EXTS	I4	3	Number of address converter extents of the Adabas file used by the store.
ADA-DA-EXTS	I4	3	Number of data extents of the Adabas file used by the store.
ADA-INDEX-LVLS	I4	3	Number of index levels in the Adabas file used by the store.
ADA-UI-PCT	I4	3	Percentage of upper index that has been used by the store.
ADA-NI-PCT	I4	3	Percentage of normal index that has been used by the store.
ADA-AC-PCT	I4	3	Percentage of address converter that has been used by the store.
ADA-DA-PCT	I4	3	Percentage of data area that has been used by the store.
TRACE-LEVEL	I2	5	PSTORE trace level.
NOTUSED0	I2	5	Alignment.

PSFCTREE-OBJECT (Struct INFO_PSTCTREE)

Field Name	Format	CIS Interface Version	Description / Action
FORMAT-VERS	I4	5	Store version format.
COLD-START-TIME	A16	5	YYYYMMDDHHMMSST cold start.
HOT-STARTS	I4	5	Hot starts since format.
MSG-DAT-FILE-NAME	A256	5	Message data file name.
MSG-DAT-FILE-SIZE	I8	5	Message data file size (64-bit).
MSG-IDX-FILE-NAME	A256	5	Message index file name.
MSG-IDX-FILE-SIZE	I8	5	Message index file size (64-bit).
STATUS-DAT-FILE-NAME	A256	5	Status data file name.
STATUS-DAT-FILE-SIZE	I8	5	Status data file size (64-bit).
STATUS-IDX-FILE-SIZE	A256	5	Status index file name.
STATUS-IDX-FILE-SIZE	I8	5	Status index file size (64-bit).
TRACE-LEVEL	I2	5	PSTORE trace level.
NOTUSED0	I2	5	Alignment.

PSFDIV-OBJECT (Struct INFO_PSFDIV)

Information services also provide detailed information on the allocation and usage of the various storage pools that implement the z/OS-DIV persistent store. This information can be used to tune the persistent store.

Note: Persistent store administration was simplified with EntireX version 9.7 and cell pool services are no longer used. Cell pool information is no longer returned in the PSFDIV-OBJECT; all returned fields from DIV-SH-IXMODULUS to DIV-CX-QUERY-RC contain zeros only.

Field Name	Format	CIS Interface Version	Description / Action
DIV-INFO-VERS	I4	2	PSD query structure version.
DIV-SH-NAME	A8	2	Persistent store name (DATASPACE-NAME in DIV-specific attributes).
DIV-SH-FORMAT-TOD	A16	2	TOD of persistent store cold start at YYYYMMDDHHMMSST.
DIV-SH-FORMAT-VERS	I4	2	Persistent store format version (PSTORE-VERSION in Broker-specific attributes).
DIV-SH-HWMARK	B4	2	Highest address in the data space.
DIV-SH-START-CNT	I4	2	Number of times the persistent store has been opened.

Field Name	Format	CIS Interface Version	Description / Action
DIV-SH-DS-ALET	B4	2	ALET (Access List Entry Token) for data space.
DIV-SH-ATT-LEN	I4	2	Length of attribute data.
DIV-SH-OID-LEN	I4	2	Length of object identifier.
DIV-SH-OID-OFF	I4	2	Offset of object identifier.
DIV-SH-IXMODULUS (1)	I4	2	Size of array/hash modulus.
DIV-SH-CP-DEF-CNT (1)	I4	2	Number of cell pool definitions.
DIV-CP-NAME (1)	A8	2	Cell pool name.
DIV-CP-CELL-SIZE (1)	I4	2	Cell size.
DIV-CP-CELL-TOTAL (1)	I4	2	Total number of cells.
DIV-CP-CELL-AVAIL (1)	I4	2	Number of cells available.
DIV-CP-EXTENT-CNT (1)	I4	2	Number of cell pool extents.
DIV-CP-QUERY-RC (1)	I4	2	Return code from cell pool query.
DIV-CX-STATUS (1)	I4	2	Cell pool extent status.
DIV-CX-EXTENT-ADDR (1)	B4	2	Address of cell pool extent.
DIV-CX-EXTENT-LEN (1)	I4	2	Length of cell pool extent.
DIV-CX-AREA-ADDR (1)	B4	2	Address of cell area.
DIV-CX-AREA-LEN (1)	I4	2	Length of cell area.
DIV-CX-CELL-TOTAL (1)	I4	2	Number of cells in extent.
DIV-CX-CELL-AVAIL (1)	I4	2	Number of cells available in extent.
DIV-CX-QUERY-RC (1)	I4	2	Return code from cell pool extent query.
TRACE-LEVEL	I2	5	PSTORE trace level.
NOTUSED0	I2	5	Alignment.
DIV-CNTLDAT-NUM-POOLS	I4	9	Number of CONTROL-DATA pools in data space.
DIV-CNTLDAT-BYTES-PER-SLOT	I4	9	Size of one slot in a CONTROL-DATA pool.
DIV-CNTLDAT-ALLOC-SLOTS	I4	9	Number of allocated slots in all CONTROL-DATA pools.
DIV-CNTLDAT-FREE-SLOTS	I4	9	Number of free slots in all CONTROL-DATA pools.
DIV-CNTLDAT-USED-SLOTS	I4	9	Number of used slots in all CONTROL-DATA pools.
DIV-LMSGDAT-NUM-POOLS	I4	9	Number of LONG-MESSAGE-DATA pools in data space.
DIV-LMSGDAT-BYTES-PER-SLOT	I4	9	Size of one slot in a LONG-MESSAGE-DATA pool.
DIV-LMSGDAT-ALLOC-SLOTS	I4	9	Number of allocated slots in all LONG-MESSAGE-DATA pools.
DIV-LMSGDAT-FREE-SLOTS	I4	9	Number of free slots in all LONG-MESSAGE-DATA pools.
DIV-LMSGDAT-USED-SLOTS	I4	9	Number of used slots in all LONG-MESSAGE-DATA pools.
DIV-SMSGDAT-NUM-POOLS	I4	9	Number of SHORT-MESSAGE-DATA pools in data space.
DIV-SMSGDAT-BYTES-PER-SLOT	I4	9	Size of one slot in a SHORT-MESSAGE-DATA pool.

Field Name	Format	CIS Interface Version	Description / Action
DIV-SMSGDAT-ALLOC-SLOTS	I4	9	Number of allocated slots in all SHORT-MESSAGE-DATA pools.
DIV-SMSGDAT-FREE-SLOTS	I4	9	Number of free slots in all SHORT-MESSAGE-DATA pools.
DIV-SMSGDAT-USED-SLOTS	I4	9	Number of used slots in all SHORT-MESSAGE-DATA pools.
DIV-UOWIDX-NUM-POOLS	I4	9	Number of UOW index pools in data space.
DIV-UOWIDX-BYTES-PER-SLOT	I4	9	Size of one slot in a UOW index pool.
DIV-UOWIDX-ALLOC-SLOTS	I4	9	Number of allocated slots in all UOW index pools.
DIV-UOWIDX-FREE-SLOTS	I4	9	Number of free slots in all UOW index pools.
DIV-UOWIDX-USED-SLOTS	I4	9	Number of used slots in all UOW index pools.

Notes:

1. Obsolete since EntireX 9.7. Returned field contains zeros only.

RESOURCE-USAGE-OBJECT (Struct INFO_RESOURCE_USAGE)

Field Name	Format	CIS Interface Version	Description / Action
TOTAL-STORAGE-ALLOCATED	I4	7	Size of allocated storage in bytes.
TOTAL-STORAGE-ALLOCATED-HIGH	I4	7	Highest size of allocated storage in bytes since Broker started.
TOTAL-STORAGE-LIMIT	I4	7	Maximum of storage that can be allocated (broker attribute MAX-MEMORY).
ACCOUNTING-BUFFERS-ALLOCATED	I4	7	ACCOUNTING: Number of buffers allocated.
ACCOUNTING-BUFFERS-FREE	I4	7	ACCOUNTING: Number of buffers free.
ACCOUNTING-BUFFERS-USED	I4	7	ACCOUNTING: Number of buffers used.
BLACKLIST-ENTRIES-ALLOCATED	I4	7	BLACKLIST: Number of entries allocated.
BLACKLIST-ENTRIES-FREE	I4	7	BLACKLIST: Number of entries free.
BLACKLIST-ENTRIES-USED	I4	7	BLACKLIST: Number of entries used.
BROKER-TO-BROKER-ENTRIES-ALLOCATED	I4	7	BROKER-TO-BROKER: Number of entries allocated.
BROKER-TO-BROKER-ENTRIES-FREE	I4	7	BROKER-TO-BROKER: Number of entries free.
BROKER-TO-BROKER-ENTRIES-USED	I4	7	BROKER-TO-BROKER: Number of entries used.
COM-BUFFERS-ALLOCATED	I4	7	COM-BUFFER: Number of buffers allocated.
COM-BUFFERS-FREE	I4	7	COM-BUFFER: Number of buffers free.
COM-BUFFERS-USED	I4	7	COM-BUFFER: Number of buffers used.
CMDLOG-FILTER-ENTRIES-ALLOCATED	I4	7	CMDLOG-FILTER: Number of entries allocated.
CMDLOG-FILTER-ENTRIES-FREE	I4	7	CMDLOG-FILTER: Number of entries free.
CMDLOG-FILTER-ENTRIES-USED	I4	7	CMDLOG-FILTER: Number of entries used.
CONNECTION-ENTRIES-ALLOCATED	I4	7	CONNECTION: Number of entries allocated.
CONNECTION-ENTRIES-FREE	I4	7	CONNECTION: Number of entries free.
CONNECTION-ENTRIES-USED	I4	7	CONNECTION: Number of entries used.
CONVERSATION-ENTRIES-ALLOCATED	I4	7	CONVERSATION: Number of entries allocated.
CONVERSATION-ENTRIES-FREE	I4	7	CONVERSATION: Number of entries free.
CONVERSATION-ENTRIES-USED	I4	7	CONVERSATION: Number of entries used.
HEAP-BYTES-ALLOCATED	I4	7	HEAP: Number of bytes allocated.
HEAP-BYTES-FREE	I4	7	HEAP: Number of bytes free.
HEAP-BYTES-USED	I4	7	HEAP: Number of bytes used.
MSG-BUFFER-LONG-ALLOCATED	I4	7	MSG-BUFFER-LONG: Number of buffers allocated.

Field Name	Format	CIS Interface Version	Description / Action
MSG-BUFFER-LONG-FREE	I4	7	MSG-BUFFER-LONG: Number of buffers free.
MSG-BUFFER-LONG-USED	I4	7	MSG-BUFFER-LONG: Number of buffers used.
MSG-BUFFER-SHORT-ALLOCATED	I4	7	MSG-BUFFER-SHORT: Number of buffers allocated.
MSG-BUFFER-SHORT-FREE	I4	7	MSG-BUFFER-SHORT: Number of buffers free.
MSG-BUFFER-SHORT-USED	I4	7	MSG-BUFFER-SHORT: Number of buffers used.
PARTICIPANT-ENTRIES-ALLOCATED	I4	7	PARTICIPANT: Number of entries allocated.
PARTICIPANT-ENTRIES-FREE	I4	7	PARTICIPANT: Number of entries free.
PARTICIPANT-ENTRIES-USED	I4	7	PARTICIPANT: Number of entries used.
PARTICIPANT-EXT-ENTRIES-ALLOCATED	I4	7	PARTICIPANT-EXT: Number of entries allocated.
PARTICIPANT-EXT-ENTRIES-FREE	I4	7	PARTICIPANT-EXT: Number of entries free.
PARTICIPANT-EXT-ENTRIES-USED	I4	7	PARTICIPANT-EXT: Number of entries used.
PROXY-QUEUE-ENTRIES-ALLOCATED	I4	7	PROXY-QUEUE: Number of entries allocated.
PROXY-QUEUE-ENTRIES-FREE	I4	7	PROXY-QUEUE: Number of entries free.
PROXY-QUEUE-ENTRIES-USED	I4	7	PROXY-QUEUE: Number of entries used.
SERVICE-ATTRIBUTE-ENTRIES-ALLOCATED	I4	7	SERVICE-ATTRIBUTE: Number of entries allocated.
SERVICE-ATTRIBUTE-ENTRIES-FREE	I4	7	SERVICE-ATTRIBUTE: Number of entries free.
SERVICE-ATTRIBUTE-ENTRIES-USED	I4	7	SERVICE-ATTRIBUTE: Number of entries used.
SERVICE-ENTRIES-ALLOCATED	I4	7	SERVICE: Number of entries allocated.
SERVICE-ENTRIES-FREE	I4	7	SERVICE: Number of entries free.
SERVICE-ENTRIES-USED	I4	7	SERVICE: Number of entries used.
SERVICE-EXT-ENTRIES-ALLOCATED	I4	7	SERVICE-EXT: Number of entries allocated.
SERVICE-EXT-ENTRIES-FREE	I4	7	SERVICE-EXT: Number of entries free.
SERVICE-EXT-ENTRIES-USED	I4	7	SERVICE-EXT: Number of entries used.
TIMEOUT-QUEUE-ENTRIES-ALLOCATED	I4	7	TIMEOUT-QUEUE: Number of entries allocated.
TIMEOUT-QUEUE-ENTRIES-FREE	I4	7	TIMEOUT-QUEUE: Number of entries free.
TIMEOUT-QUEUE-ENTRIES-USED	I4	7	TIMEOUT-QUEUE: Number of entries used.
TRANSLATION-ENTRIES-ALLOCATED	I4	7	TRANSLATION: Number of entries allocated.
TRANSLATION-ENTRIES-FREE	I4	7	TRANSLATION: Number of entries free.
TRANSLATION-ENTRIES-USED	I4	7	TRANSLATION: Number of entries used.

Field Name	Format	CIS Interface Version	Description / Action
UOW-ENTRIES-ALLOCATED	I4	7	UOW: Number of entries allocated.
UOW-ENTRIES-FREE	I4	7	UOW: Number of entries free.
UOW-ENTRIES-USED	I4	7	UOW: Number of entries used.
WORK-QUEUE-ENTRIES-ALLOCATED	I4	7	WORK-QUEUE: Number of entries allocated.
WORK-QUEUE-ENTRIES-FREE	I4	7	WORK-QUEUE: Number of entries free.
WORK-QUEUE-ENTRIES-USED	I4	7	WORK-QUEUE: Number of entries used.

SECURITY-OBJECT (Struct INFO_SECURITY)

Field Name	Format	CIS Interface Version	Description / Action
COUNT-AUTHENTICATION-SUCCEEDED	I4	5	Successful authentications.
COUNT-AUTHENTICATION-FAILED	I4	5	Failed authentications.
COUNT-AUTHORIZATION-SUCCEEDED	I4	5	Successful authorizations.
COUNT-AUTHORIZATION-FAILED	I4	5	Failed authorizations.
SAF-PROFILE-LENGTH	I4	5	Max profile length (CDT) m/f.
TRACE-LEVEL	I2	5	Security trace level.
SECURITY-LEVEL	I2	5	Security Level m/f.
AUTHENTICATION-TYPE	A8	5	Authentication type.
SAF-CLASS	A8	5	SAF profile CLASS (8) m/f.
SECURITY-NODE	A8	5	Security node m/f.
INCLUDE-CLASS	I1	5	Include CLASS in prof m/f.
INCLUDE-NAME	I1	5	Include NAME in prof m/f.
INCLUDE-SERVICE	I1	5	Include SERVICE in prof m/f.
UNIVERSAL	I1	5	Allow undefined profile m/f.
CHECK-IP-ADDRESS	I1	5	Check IP address m/f.
WARN-MODE	I1	5	Run in warn mode m/f.
IGNORE-STOKEN	I1	5	Ignore ACI STOKEN m/f.
TRUSTED-USER	I1	5	Trusted User ID m/f.
PROPAGATE-TRUSTED-USER	I1	5	VerifiedId m/f.
PASSWORD-TO-UPPER-CASE	I1	5	Convert password to uppercase m/f.
NOTUSED0	A2	5	Alignment.

SERVICE-OBJECT (Struct INFO_SV)

Field Name	Format	CIS Interface Version	Description / Action
SERVER-CLASS	A32	1	Name of server class.
SERVER-NAME	A32	1	Name of server.
SERVICE	A32	1	Name of service. The header contains the maximum length for the SERVER-CLASS, SERVER-NAME and SERVICE fields for all retrieved objects. See *Common Header Structure for Response Data*.
TRANS	A8	1	Name of translation routine used.
CONV-NONACT	I4	1	Conversation timeout (corresponds to CONV-NONACT for the service in the attribute file).
SERVER-ACT	I4	1	Number of servers active for service. This counter also includes the active Attach Server instances. You may subtract the field ATTACH-MGRS-ACT to calculate the number of ordinary server instances.
CONV-ACT	I4	1	Number of conversations active for service.
CONV-HIGH	I4	1	Highest number of conversations active for service.
LONG-ACT	I4	1	Number of long buffers active (in use) for the service.
LONG-HIGH	I4	1	Highest number of long buffers active (in use) for the service.
SHORT-ACT	I4	1	Number of short buffers active (in use) for the service.
SHORT-HIGH	I4	1	Highest number of short buffers active (in use) for the service.
NUM-WAIT-SERVER	I4	1	Number of times a client had to wait for this service or messages from the server.
NUM-SERV-OCC	I4	1	Number of times a client request (SEND with CONVID=NEW or NONE) could not be immediately assigned to a waiting server, i.e. all servers offering this service are occupied.
NUM-PEND	I4	1	Number of new conversations which are currently in the queue, but not yet assigned to a server (pending).
PEND-HIGH	I4	1	Highest number of pending conversations.
REQ-SUM	I4	1	Accumulated number of requests (number of SEND commands with CONVID=NEW or NONE).
LMAXUOWS	I4	2	Maximum number of active UOWs
LMAXUOWMSG	I4	2	Maximum number of messages in a UOW
LUWTIME	I4	2	Maximum UOW lifetime
LMAXDELCNT	I4	2	Is currently not in use (count is always zero.)
LMAXMSGSIZE	I4	2	Maximum size of a message
LTOTALUOWS	I4	2	Number of UOWs

Field Name	Format	CIS Interface Version	Description / Action
CSTORE	I1	2	Store attribute for all UOWs: 0=OFF 1=BROKER
CUWSTATP	I1	2	UOWstatus lifetime multiplier (0-255)
CDEFERRED	I1	2	Default status attribute for all UOWs: 0 NO 1 YES
CENCLEVEL	I1	3	Deprecated. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.
ATTACH-MGRS-ACT	I4	4	Number of attach servers active
LUWSTAT-ADD-TIME	I4	4	Unit of work status additional lifetime
NUM-CONV	I4	5	Number of conversations.
NUM-SERVER	I4	5	Number of servers.
NUM-LONG-MSG-BUFFER	I4	5	Number of long message buffers.
NUM-SHORT-MSG-BUFFER	I4	5	Number of short message buffers.
CONVERSION	A8	5	Name of conversion routine.
CONVERSION-PARMS	A255	5	Conversion parameters.
NOTUSED1	A1	5	Alignment.
RESERVED	I4	5	Reserved for future use.
ARF	I1	10	0 Ordinary service. 1 Service belongs to Adabas Event Replication.
SCM	I1	10	If ARF=1: 0 Single Conversation Mode was not activated for this service. 1 Single Conversation Mode was activated for this service.
PREFETCH	I1	10	If ARF=1 and SCM=1: 0 Server is not prefetching units of work 1 Server is prefetching units of work
RPC	A5	10	YES Server identified as RPC server. NO Server identified as ACI server. UNDEF Server not identified at all.

Field Name	Format	CIS Interface Version	Description / Action
MAX-POSTPONE-ATTEMPTS	I4	10	Number of POSTPONE-ATTEMPTS configured for this service.
POSTPONE-DELAY	I4	10	POSTPONE-DELAY in number of seconds configured for this service.

SSL-OBJECT (Struct INFO_SSL)

Field Name	Format	CIS Interface Version	Description / Action
CLONE-INDEX	I4	5	Clone index.
STATUS	I4	5	Status of communicator. Possible values defined as ETB_INFO_COM_STATUS_.
OPEN-CONNECTIONS	I4	5	Number of open connections.
MAX-CONNECTIONS	I4	5	Maximum number of connections.
PORT-NUMBER	I4	5	Port number.
IP-ADDRESS	A16	6	IPv4 address corresponding to attribute HOST in DEFAULTS=SSL section of Broker attribute file.
HOST-NAME	A256	6	Host name specified using attribute HOST in DEFAULTS=SSL section of Broker attribute file.
TASK-RUNNING	I1	8	Transport task running. 0=NO, 1=YES.
IPV6-ADDRESS	A46	8	IPV6 address corresponding to attribute HOST in the DEFAULTS=SSL section of the Broker attribute file.

STATISTICS-OBJECT (Struct INFO_STATISTICS) (Excerpt of BROKER-OBJECT)

Field Name	Format	CIS Interface Version	Description / Action
NUM-SERVICE	I4	7	Number of services defined (see NUM-SERVER).
SERVICE-ACT	I4	7	Number of services active.
NUM-CLIENT	I4	7	Number of clients defined (see NUM-CLIENT).
CLIENT-ACT	I4	7	Number of clients active.
CLIENT-HIGH	I4	7	Highest number of clients active since Broker started.
NUM-SERVER	I4	7	Number of servers (see NUM-SERVER).
SERVER-ACT	I4	7	Number of servers active. This counter also includes the active Attach Server instances.
SERVER-HIGH	I4	7	Highest number of servers active since Broker started.
NUM-CONV	I4	7	Number of conversations defined (see NUM-CONVERSATION).
CONV-ACT	I4	7	Number of conversations active.
CONV-HIGH	I4	7	Highest number of conversations active since Broker started.
NUM-LONG	I4	7	Number of long buffers defined (see NUM-LONG-BUFFER).
LONG-ACT	I4	7	Number of long buffers active.
LONG-HIGH	I4	7	Highest number of long buffers active since Broker started.
NUM-SHORT	I4	7	Number of short buffers defined (see NUM-SHORT-BUFFER).

Field Name	Format	CIS Interface Version	Description / Action
SHORT-ACT	I4	7	Number of short buffers active.
SHORT-HIGH	I4	7	Highest number of short buffers active since Broker started.

TCP-OBJECT (Struct INFO_TCP)

Field Name	Format	CIS Interface Version	Description / Action
CLONE-INDEX	I4	5	Clone index.
STATUS	I4	5	Status of communicator ETB_INFO_COM_STATUS_.
OPEN-CONNECTIONS	I4	5	Number of open connections.
MAX-CONNECTIONS	I4	5	Maximum number of connections. Platform-dependent. See *Maximum TCP/IP Connections per Communicator* under *Broker Resource Allocation*.
PORT-NUMBER	I4	5	Port number.
IP-ADDRESS	A16	6	IPv4 address corresponding to attribute HOST in DEFAULTS=TCP section of Broker attribute file.
HOST-NAME	A256	6	Host name specified using attribute HOST in DEFAULTS=TCP section of Broker attribute file.
TASK-RUNNING	I1	8	Transport task running. 0=NO, 1=YES.
IPV6-ADDRESS	A46	8	IPV6 address corresponding to attribute HOST in the DEFAULTS=TCP section of the Broker attribute file.

UOW-STATISTICS (Struct INFO_UOW_STATISTICS)

Field Name	Format	CIS Interface Version	Description / Action
SERVER-CLASS	A32	9	Name of server class.
SERVER-NAME	A32	9	Name of server.
SERVICE	A32	9	Name of service.
CURRENT-NUMBER-OF-UOWS	I8	9	Current number of units of work for this service.
CURRENT-NUMBER-OF-UOW-MESSAGES	I8	9	Current number of UOW messages for this service.
CURRENT-NUMBER-OF-BYTES-IN-ALL-UOW-MESSAGES	I8	9	Current number of bytes in all UOW messages for this service.

Field Name	Format	CIS Interface Version	Description / Action
MAX-VALUE-MESSAGES-IN-ONE-UOW	I4	9	Max value of 'number of messages in one UOW'.
MAX-VALUE-NUMBER-OF-BYTES-IN-ONE-UOW	I4	9	Max value of 'number of bytes in one UOW'.
OLDEST-TIME-OF-UOW-CREATION	A32	9	Date and time of the oldest UOW for this service.
NEWEST-TIME-OF-UOW-CREATION	A32	9	Date and time of the newest UOW for this service.

USER-OBJECT (Struct INFO_USER)

Field Name	Format	CIS Interface Version	Description / Action
USER-ID	A32	7	Corresponds to USER-ID in the ACI. The maximum length of this field is determined by field MAX-UID-LEN in the header. See Common Header Structure.
IS-CLIENT	I1	7	Is user a client? 0 NO 1 YES
IS-SERVER	I1	7	Is user a server? 0 NO 1 YES
SEQNO	I4	7	Unique sequence number of user. Can be used with CIS command SHUTDOWN.
ENDIAN	I2	7	Endian type of user's platform: 1 BIG ENDIAN (high order first) 0 LITTLE ENDIAN
CHAR-SET	I2	7	Character set of user's platform: 34 EBCDIC IBM 66 EBCDIC SNI 1 ASCII PC 386 16 ASCII PC OS/2 128 ASCII 8859-1

Field Name	Format	CIS Interface Version	Description / Action
P-USER-ID	B28	7	Specifies the physical internal unique ID which is used to distinguish between several users with the same user ID. This field is used as a handle, i.e. no translation is performed. With CIS commands SHUTDOWN PARTICIPANT and SHUTDOWN SERVER, field SEQNO is provided as unique criterion.
TOKEN	A32	7	Corresponds to TOKEN in the ACI. The maximum length of this field is determined by MAX-TK-LEN in the header. See *Common Header Structure for Response Data*.
LAST-ACTIVE	I4	7	Elapsed time since the last activity of the user.

WORKER-OBJECT (Struct INFO_WKR)

Field Name	Format	CIS Interface Version	Description / Action
WORKER-ID	I2	1	The worker ID is the table number of this worker's worker queue entry.
WORKER-STAT	I2	1	Status of worker: 2 ACTIVE 4 STARTED 5 WAITING
CALL-SUM	I4	1	Sum of calls per worker since Broker started.
IDLE-SUM	I4	1	Sum of idle time per worker since Broker started.

WORKER-USAGE-OBJECT (Struct INFO_WORKER-USAGE)

Field Name	Format	CIS Interface Version	Description / Action
WORKER-MAX-ATTRIBUTE	I4	7	Maximum number of worker tasks the Broker can use. See broker attribute WORKER-MAX.
WORKER-MIN-ATTRIBUTE	I4	7	Minimum number of worker tasks the Broker can use. See broker attribute WORKER-MIN.
WORKER-NONACT-ATTRIBUTE	I4	7	Non-activity time in seconds to elapse before a worker tasks is stopped. See broker attribute WORKER-NONACT.

Field Name	Format	CIS Interface Version	Description / Action
WORKER-QUEUE-DEPTH	I4	7	Number of unassigned user requests in the input queue before another worker task gets started. See broker attribute WORKER-QUEUE-DEPTH.
WORKER-START-DELAY-ATTRIBUTE	I4	7	Delay after a successful worker task invocation before another worker task can be started. See broker attribute WORKER-START-DELAY.
LAST-START-TIME	I4	7	Time of last worker startup.
LAST-STOP-TIME	I4	7	Time of last worker stop.
EFFECTIVE-START-DELAY-SECONDS	I4	7	Time value representing the seconds of the timeval structure that contains the effective time consumption starting a worker task.
EFFECTIVE-START-DELAY-MICRO-SECONDS	I4	7	Time value representing the microseconds of the timeval structure that contains the effective time consumption starting a worker task.
WORKER-HIGH	I4	7	Highest number of worker tasks active since Broker started.
WORKER-LOW	I4	7	Lowest number of worker tasks active since Broker started.