webMethods EntireX

ACI for C

Version 9.5 SP1

November 2013

This document applies to webMethods EntireX Version 9.5 SP1.

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

Copyright © 1997-2013 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://documentation.softwareag.com/legal/.

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://documentation.softwareag.com/legal/ 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 and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: EXX-ACI-95SP1-20140628C

Table of Contents

1 EntireX Broker ACI for C

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 C.

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`.

You can set the send buffer and the receive buffer to null if they are not required by the selected EntireX Broker function.

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

- Under all platforms and with all Broker stubs see the prototype. For example:

  ```
  int broker (ETBCB *, char *, char *, char *);
  rc = broker( pCb, pSBuf, pRBuf, pEBuf );
  ```

- additionally, under z/OS, you can invoke *CICSETB* under *Administration of Broker Stubs under z/OS*, using the following `EXEC CICS LINK` command. The length of the COMMAREA is always 24. For example:

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

  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 EntireX 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 the Broker stub is used as a function, the stub returns the last four bytes of the `ERROR-CODE` field in the EntireX Broker control block, i.e. the error number.

If 0 (zeros) are returned in the `ERROR-CODE` field in all positions of the character array, the operation has been performed successfully. However, function results other than 0 (zeros) in all positions do not necessarily indicate an error. See *Error Handling* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.

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* in the ACI Programming documentation. See the actual layout for C in ***Broker ACI Control Block Definition*** below.

Note: Header files and examples are provided as models if you want to write your own ACI appliations (see ***ACI Examples and Header Files*** 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.

Broker ACI Field	C Definition	Description / Related Information		API Vers.	Notes
`API-TYPE`	`ETB_BYTE api_type`	API type	See *`API-TYPE` and `API-VERSION`* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.	1	
`API-VERSION`	`ETB_BYTE api_version`	API version.		1	
`FUNCTION`	`ETB_BYTE function`	See *Broker ACI Fields* in the ACI Programming documentation.		1	
`OPTION`	`ETB_BYTE option`	See `OPTION` under *Broker ACI Fields*.		1	
	`ETB_CHAR reserved1[16]`	Reserved for future use.		1	1
`SEND-LENGTH`	`ETB_LONG send_length`	Send length	See *Using Send and Receive Buffers* under *Writing Applications:*	1	
`RECEIVE-LENGTH`	`ETB_LONG receive_length`	Receive length.		1	
`RETURN-LENGTH`	`ETB_LONG return_length`	Return length.		1	

Broker ACI Field	C Definition	Description / Related Information	API Vers.	Notes
		Client and Server | Publish and Subscribe in the ACI Programming documentation.		
`ERRTEXT-LENGTH`	`ETB_LONG errtext_length`	Error text length.	1	
`BROKER-ID`	`ETB_CHAR broker_id[S_BROKER_ID]`	Broker ID. See *Using the Broker ID in Applications* in the ACI Programming documentation.	1	

`SERVER-CLASS`
`SERVER-NAME`
`SERVICE`
	`ETB_CHAR server_class[S_SERVER-CLASS]`
	`ETB_CHAR server_name[S_SERVER-NAME]`
	`ETB_CHAR service[S_SERVICE]`
		Service. See *Control Block Fields and Verbs* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.	1	3, 5

`USER-ID`	`ETB_CHAR user_id[S_USER_ID]`	User ID. See *`USER-ID` and `TOKEN`* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.	1	
`PASSWORD`	`ETB_CHAR password[S_PASSWORD]`	Password. See *Authentication* under *Writing Applications using EntireX Security* in the ACI Programming documentation.	1	4,5
`TOKEN`	`ETB_CHAR token[S_TOKEN]`	Reconnection token. See *`USER-ID` and `TOKEN`* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.	1	3, 5
`SECURITY-TOKEN`	`ETB_CHAR security_token[S_securityToken]`	Security token. See *Writing Applications using EntireX Security* in the ACI Programming documentation.	1	4, 5
`CONV-ID`	`ETB_CHAR conv_id[S_CONV_ID]`	Conversation ID. See *Conversational and Non-conversational Mode* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.	1	3, 5
`WAIT`	`ETB_CHAR wait[S_WAIT]`	Wait value. See *Blocked and Non-blocked Broker Calls* under	1	3, 5

Broker ACI Field	C Definition	Description / Related Information	API Vers.	Notes
		Writing Applications: Client and Server | *Publish and Subscribe* in the ACI Programming documentation.		
`ERROR-CODE`	`ETB_CHAR error_code[S_ERROR_CODE]`	Error code. See *Error Handling* under *Writing Applications: Client and Server* | *Publish and Subscribe* in the ACI Programming documentation.	1	
`ENVIRONMENT`	`ETB_CHAR environment[S_ENVIRONMENT]`	Environment. See *Using Internationalization* under *Writing Applications: Client and Server* | *Publish and Subscribe* in the ACI Programming documentation.	1	3, 5
`ADCOUNT`	`ETB_LONG adcount`	Attempted delivery count. See *Writing Applications: Units of Work* in the ACI Programming documentation.	2	
`USER-DATA`	`ETB_CHAR user_data[S_USERDATA]`	Conversation User Data. See *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.	2	3, 5
	`ETB_CHAR ptime[S_PTIME]`	Reserved for future use.	2	1, 3, 5
`NEWPASSWORD`	`ETB_CHAR newpassword[S_PASSWORD]`	New password. See *Writing Applications using EntireX Security* in the ACI Programming documentation.	2	4, 5
`CLIENT-UID`	`ETB_CHAR client_uid[S_CLIENTUID]`	Client User ID. See *Client User ID* under *Writing Applications using EntireX Security* in the ACI Programming documentation.	2	
`CONV-STAT`	`ETB_BYTE conv_stat`	Conversation status. See *Conversational and Non-conversational Mode* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.	2	
`STORE`	`ETB_BYTE store`	Persistence or non-persistence of a UOW. See *Writing*	2	

Broker ACI Field	C Definition	Description / Related Information		API Vers.	Notes
		Applications: Units of Work in the ACI Programming documentation.			
	`ETB_BYTE status`	Reserved for future use.		2	1
`UOWSTATUS`	`ETB_BYTE uowStatus`	UOW Status.	See *Writing Applications: Units of Work* in the ACI Programming documentation.	3	3, 5
`UWTIME`	`ETB_CHAR uowTime[S_WAIT]`	UOW lifetime.		3	3, 5
`UOWID`	`ETB_CHAR uowID[S_UOW_ID]`	UOW unique identifier.		3	3, 5
`USTATUS`	`ETB_CHAR userStatus[S_U_STATUS]`	User status.		3	
`UOW-STATUS-PERSIST`	`ETB_BYTE uowStatusPersist`	Multiplier for persistent status lifetime.		3	2
	`ETB_CHAR reserved2[3]`	Reserved for future use.		3	1
`LOCALE-STRING`	`ETB_CHAR locale_string[S_LOCALE]`	Locale string. To be used to override or provide codepages. See *Using Internationalization* under *Writing Applications: Client and Server* | *Publish and Subscribe* in the ACI Programming documentation.		4	
`DATA-ARCH`	`ETB_BYTE data_arch`	Data architecture. See *ICU Conversion* under *Introduction to Internationalization*.		4	2
`FORCE-LOGON`	`ETB_CHAR forceLogon`	Override Broker attribute `AUTOLOGON`.	See *Writing Applications using EntireX Security* in the ACI Programming documentation	6	
`ENCRYPTION-LEVEL`	`ETB_BYTE encryptionLevel`	Encryption level.		6	2
`KERNELSECURITY`	`ETB_CHAR kernelsecurity`	Kernel security.		7	
`COMMITTIME`	`ETB_CHAR commitTime[S_COMMIT_TIME]`	Commit time. See *Writing Applications: Units of Work* in the ACI Programming documentation.		7	
`COMPRESSLEVEL`	`ETB_CHAR compress`	Compression level. See *Data Compression* under *Writing Applications: Client and Server* | *Publish and Subscribe* in the ACI Programming documentation.		7	

<table>
<tr><th>Broker ACI Field</th><th>C Definition</th><th colspan="2">Description /
Related Information</th><th>API Vers.</th><th>Notes</th></tr>
<tr><td></td><td><code>ETB_BYTE reserved3[2]</code></td><td colspan="2">Reserved for future use.</td><td>8</td><td>1</td></tr>
<tr><td></td><td><code>ETB_LONG reserved4</code></td><td colspan="2">Reserved for future use.</td><td>8</td><td>1</td></tr>
<tr><td><code>UWSTAT-LIFETIME</code></td><td><code>ETB_CHAR uwStatLifeTime[S_WAIT]</code></td><td colspan="2">Add value for persistent status lifetime. See broker attribute <code>UWSTAT-LIFETIME</code>.</td><td>8</td><td></td></tr>
<tr><td><code>TOPIC</code></td><td><code>ETB_CHAR topic[S_TOPIC]</code></td><td>Topic name for publish and subscribe.</td><td rowspan="2">See <i>Writing Applications: Publish and Subscribe</i> in the ACI Programming documentation.</td><td>8</td><td></td></tr>
<tr><td><code>PUBLICATION-ID</code></td><td><code>ETB_CHAR publicationID[S_PUBLICATION_ID]</code></td><td>Publication ID for publish and subscribe.</td><td>8</td><td></td></tr>
<tr><td><code>CLIENT-ID</code></td><td><code>ETB_LONG client_id</code></td><td colspan="2">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).</td><td>9</td><td></td></tr>
<tr><td><code>LOG-COMMAND</code></td><td><code>ETB_BYTE logCommand</code></td><td colspan="2">Log the current command. See also <i>Programmatically Turning on Command Logging</i> under <i>Writing Applications: Client and Server | Publish and Subscribe</i> in the ACI Programming documentation</td><td>9</td><td></td></tr>
<tr><td><code>CREDENTIALS-TYPE</code></td><td><code>ETB_BYTE credentialsType</code></td><td colspan="2">Indicates the credentials type to be used to authenticate a user. The default is to use user ID and password. Enter "1" to specify IAF authentication. See also <i>IAF Authentication</i> under <i>Writing Applications: Client and Server | Publish and Subscribe</i> in the ACI Programming documentation.</td><td>9</td><td></td></tr>
<tr><td><code>VARLIST-OFFSET</code></td><td><code>ETB_LONG varlist_offset</code></td><td colspan="2">Internal Software AG field.</td><td>10</td><td></td></tr>
<tr><td><code>LONG-BROKER-ID-LENGTH</code></td><td><code>ETB_LONG LONG-BROKER-ID-LENGTH</code></td><td colspan="2">See <code>LONG-BROKER-ID-LENGTH</code>.</td><td>10</td><td></td></tr>
</table>

Notes:

1. Reserved for future use.

2. You must set this field to a low value (0x00) if you do not intend to use it.
3. The field is transmitted up to the first blank or low value (0x00). It is not transmitted if the first character is a blank or a low value (0x00).
4. All trailing low values (0x00) are truncated. The field is not transmitted if the entire field is a low value (0x00).
5. If fields are not needed for a specific command function, suppress their transmission by initializing them to blanks or low value (0x00).

Broker ACI Control Block Definition

The Developer's Kit provides a header file with the ACI control block definition. See ***ACI Examples and Header Files*** for where it is provided on your platform.

```
/*
***************************************************************************
* Product        : EntireX Broker
* Copyright      : (c) Copyright Software AG 1997 - 2012. All rights reserved.
* Version        : 9.0
* File           : ETBCDEF.H
* File Version   : $Revision: 1.2 $
* Description    : C language ACI control block definitions.
*
*/

#ifndef ETBCDEF__H_
#define ETBCDEF__H_

#ifdef __cplusplus
extern "C" {
#endif

/* --- Type Definitions --------------------------------------------- */

#define ETB_BYTE  unsigned char  /* 1 byte unsigned integer           */
#define ETB_CHAR  char           /* 1 byte character                  */
#define ETB_LONG  int            /* 4 byte signed integer             */
#define ETB_SHORT short          /* 2 byte signed integer             */

/* --- EntireX Broker API Type Constants (api_type) ----------------- */

#define API_TYPE1                      ((ETB_BYTE) 0x01)
#define API_TYPE2                      ((ETB_BYTE) 0x02)
#define API_TYPE4                      ((ETB_BYTE) 0x04)
#define API_TYPE8                      ((ETB_BYTE) 0x08)
```

```
/* --- EntireX Broker API Version Constants (api_version) ----------- */

#define API_VERS1                      ((ETB_BYTE) 1)
#define API_VERS2                      ((ETB_BYTE) 2)
#define API_VERS3                      ((ETB_BYTE) 3)
#define API_VERS4                      ((ETB_BYTE) 4)
#define API_VERS5                      ((ETB_BYTE) 5)
#define API_VERS6                      ((ETB_BYTE) 6)
#define API_VERS7                      ((ETB_BYTE) 7)
#define API_VERS8                      ((ETB_BYTE) 8)
#define API_VERS9                      ((ETB_BYTE) 9)
#define API_VERS10                     ((ETB_BYTE) 10)
#define API_VERS_HIGHEST               API_VERS10

/* --- EntireX Broker API API Function Constants (function) --------- */

#define FCT_SEND                       ((ETB_BYTE) 1)
#define FCT_RECEIVE                    ((ETB_BYTE) 2)
#define FCT_UNDO                       ((ETB_BYTE) 4)
#define FCT_EOC                        ((ETB_BYTE) 5)
#define FCT_REGISTER                   ((ETB_BYTE) 6)
#define FCT_DEREGISTER                 ((ETB_BYTE) 7)
#define FCT_VERSION                    ((ETB_BYTE) 8)
#define FCT_LOGON                      ((ETB_BYTE) 9)
#define FCT_LOGOFF                     ((ETB_BYTE) 10)
#define FCT_SET                        ((ETB_BYTE) 11)
#define FCT_GET                        ((ETB_BYTE) 12)
#define FCT_SYNCPOINT                  ((ETB_BYTE) 13)
#define FCT_KERNELVERS                 ((ETB_BYTE) 14)
#define FCT_LOCTRANS                   ((ETB_BYTE) 15)
#define FCT_SETSSLPARMS                ((ETB_BYTE) 16)
#define FCT_SEND_PUBLICATION           ((ETB_BYTE) 17)
#define FCT_RECEIVE_PUBLICATION        ((ETB_BYTE) 18)
#define FCT_SUBSCRIBE                  ((ETB_BYTE) 19)
#define FCT_UNSUBSCRIBE                ((ETB_BYTE) 20)
#define FCT_CONTROL_PUBLICATION        ((ETB_BYTE) 21)
#define FCT_REPLY_ERROR                ((ETB_BYTE) 22)

/* --- EntireX Broker API Option Constants (option) ----------------- */

#define OPT_OFF                        ((ETB_BYTE) 0)
#define OPT_MSG                        ((ETB_BYTE) 1)
#define OPT_HOLD                       ((ETB_BYTE) 2)
#define OPT_IMMED                      ((ETB_BYTE) 3)
#define OPT_QUIESCE                    ((ETB_BYTE) 4)
#define OPT_EOC                        ((ETB_BYTE) 5)
#define OPT_CANCEL                     ((ETB_BYTE) 6)
#define OPT_LAST                       ((ETB_BYTE) 7)
#define OPT_NEXT                       ((ETB_BYTE) 8)
#define OPT_PREVIEW                    ((ETB_BYTE) 9)
#define OPT_COMMIT                     ((ETB_BYTE) 10)
#define OPT_BACKOUT                    ((ETB_BYTE) 11)
```

```
#define OPT_SYNC                       ((ETB_BYTE) 12)
#define OPT_ATTACH                     ((ETB_BYTE) 13)
#define OPT_DELETE                     ((ETB_BYTE) 14)
#define OPT_EOCCANCEL                  ((ETB_BYTE) 15)
#define OPT_QUERY                      ((ETB_BYTE) 16)
#define OPT_SETUSTATUS                 ((ETB_BYTE) 17)
#define OPT_ANY                        ((ETB_BYTE) 18)
#define OPT_TERMINATE                  ((ETB_BYTE) 19)
#define OPT_DURABLE                    ((ETB_BYTE) 20)
#define OPT_CHECKSERVICE               ((ETB_BYTE) 21)

/* --- EntireX Broker Environment Constants (environment) ----------- */

#define ETB_ENVIRONMENT_NO_CONVERSION  "NONE"

/* --- EntireX Broker Conversation Status Constants (conv_stat) ----- */

#define CONVSTAT_NEW                   ((ETB_BYTE) 1)
#define CONVSTAT_OLD                   ((ETB_BYTE) 2)
#define CONVSTAT_NONE                  ((ETB_BYTE) 3)

/* --- EntireX Broker Store Constants (store) ----------------------- */

#define STORE_OFF                      ((ETB_BYTE) 1)
#define STORE_BROKER                   ((ETB_BYTE) 2)

/* --- EntireX Broker Status Constants (status) --------------------- */

#define STAT_OFF                       ((ETB_BYTE) 1)
#define STAT_STORED                    ((ETB_BYTE) 2)
#define STAT_DELIVERY_ATTEMP           ((ETB_BYTE) 3)
#define STAT_DELIVERED                 ((ETB_BYTE) 4)
#define STAT_PROCESSED                 ((ETB_BYTE) 5)
#define STAT_DEAD                      ((ETB_BYTE) 6)

/* --- EntireX Broker UOW Status Constants (uowStatus) -------------- */

#define RECV_NONE                      ((ETB_BYTE) 0)
#define RECEIVED                       ((ETB_BYTE) 1)
#define ACCEPTED                       ((ETB_BYTE) 2)
#define DELIVERED                      ((ETB_BYTE) 3)
#define BACKEDOUT                      ((ETB_BYTE) 4)
#define PROCESSED                      ((ETB_BYTE) 5)
#define CANCELLED                      ((ETB_BYTE) 6)
#define TIMEOUT                        ((ETB_BYTE) 7)
#define DISCARDED                      ((ETB_BYTE) 8)
#define RECV_FIRST                     ((ETB_BYTE) 9)
#define RECV_MIDDLE                    ((ETB_BYTE) 10)
#define RECV_LAST                      ((ETB_BYTE) 11)
#define RECV_ONLY                      ((ETB_BYTE) 12)

/* --- EntireX Broker Locale String Constants (locale_string) ------- */
```

```
#define ETB_CODEPAGE_USE_PLATFORM_DEFAULT "LOCAL"

/* --- EntireX Broker Architecture Constants (data_arch) ------------ */

#define ACODE_HIGH_ASCII_IBM           ((ETB_BYTE) 1)
#define ACODE_LOW__ASCII_IBM           ((ETB_BYTE) 2)
#define ACODE_HIGH_EBCDIC_IBM          ((ETB_BYTE) 3)
#define ACODE_LOW__EBCDIC_IBM          ((ETB_BYTE) 4)
#define ACODE_HIGH_ASCII_VAX           ((ETB_BYTE) 5)
#define ACODE_LOW__ASCII_VAX           ((ETB_BYTE) 6)
#define ACODE_HIGH_EBCDIC_VAX          ((ETB_BYTE) 7)
#define ACODE_LOW__EBCDIC_VAX          ((ETB_BYTE) 8)
#define ACODE_HIGH_ASCII_IEEE          ((ETB_BYTE) 9)
#define ACODE_LOW__ASCII_IEEE          ((ETB_BYTE) 10)
#define ACODE_HIGH_EBCDIC_IEEE         ((ETB_BYTE) 11)
#define ACODE_LOW__EBCDIC_IEEE         ((ETB_BYTE) 12)
#define ACODE_HIGHEST_VALUE            ((ETB_BYTE) 12)

/* --- EntireX Broker Force Logon Constants (forceLogon) ------------ */

#define FORCE_LOGON_NO                 ((ETB_CHAR) 'N')
#define FORCE_LOGON_YES                ((ETB_CHAR) 'Y')
#define FORCE_LOGON_S                  ((ETB_CHAR) 'S')

/* --- EntireX Broker Encryption Level Constants (encryptionLevel) -- */

#define ENCLEVEL_NONE                  ((ETB_BYTE) 0)
#define ENCLEVEL_TO_BROKER             ((ETB_BYTE) 1)
#define ENCLEVEL_TO_TARGET             ((ETB_BYTE) 2)

/* --- EntireX Broker Kernel Security Constants (kernelSecurity) ---- */

#define KERNEL_SECURITY_NO             ((ETB_CHAR) 'N')
#define KERNEL_SECURITY_YES            ((ETB_CHAR) 'Y')
#define KERNEL_SECURITY_USER           ((ETB_CHAR) 'U')
#define KERNEL_SECURITY_LIGHT          ((ETB_CHAR) 'L')

/* --- EntireX Broker Compression Level Constants (compress) -------- */

#define COMPRESS_LEVEL_0               ((ETB_CHAR) '0')
#define COMPRESS_LEVEL_1               ((ETB_CHAR) '1')
#define COMPRESS_LEVEL_2               ((ETB_CHAR) '2')
#define COMPRESS_LEVEL_3               ((ETB_CHAR) '3')
#define COMPRESS_LEVEL_4               ((ETB_CHAR) '4')
#define COMPRESS_LEVEL_5               ((ETB_CHAR) '5')
#define COMPRESS_LEVEL_6               ((ETB_CHAR) '6')
#define COMPRESS_LEVEL_7               ((ETB_CHAR) '7')
#define COMPRESS_LEVEL_8               ((ETB_CHAR) '8')
#define COMPRESS_LEVEL_9               ((ETB_CHAR) '9')
#define COMPRESS_LEVEL_NO              ((ETB_CHAR) 'N')
#define COMPRESS_LEVEL_YES             ((ETB_CHAR) 'Y')
```

```
/* --- EntireX Broker Credentials Type Constants (credentialsType) -- */

#define CREDENTIALS_TYPE_UID_PWD       ((ETB_BYTE) 0)
#define CREDENTIALS_TYPE_IAF           ((ETB_BYTE) 1)

/*--------------------------------------------------------------------*/
/* The first 4 bytes of the first reserved field (reserved) can be    */
/* used to specify a stub trace level                                 */
/*--------------------------------------------------------------------*/

#define STUBLOG_EYECATCHER_ARRAY    'T', 'L', '='
#define STUBLOG_OFF                 '0'
#define STUBLOG_LEVEL0              STUBLOG_OFF
#define STUBLOG_LEVEL1              '1'
#define STUBLOG_LEVEL2              '2'
#define STUBLOG_LEVEL3              '3'
#define STUBLOG_LEVEL4              '4'

/* --- EntireX Broker API Size of fields ---------------------------- */

#define S_ADAPTERR                     ((ETB_CHAR) 8)
#define S_APPLICATION_NAME             ((ETB_CHAR) 64)
#define S_APPLICATION_TYPE             ((ETB_CHAR) 8)
#define S_BROKER_ID                    ((ETB_CHAR) 32)
#define S_CLIENTUID                    ((ETB_CHAR) 32)
#define S_COMMIT_TIME                  ((ETB_CHAR) 17)
#define S_CONV_ID                      ((ETB_CHAR) 16)
#define S_ENVIRONMENT                  ((ETB_CHAR) 32)
#define S_ERROR_CODE                   ((ETB_CHAR) 8)
#define S_ERROR_CLASS                  ((ETB_CHAR) 4)
#define S_ERROR_NUMBER                 ((ETB_CHAR) 4)
#define S_LOCALE                       ((ETB_CHAR) 40)
#define S_MSGID                        ((ETB_CHAR) 32)
#define S_MSGTYPE                      ((ETB_CHAR) 16)
#define S_NODENAME                     ((ETB_CHAR) 32)
#define S_PASSWORD                     ((ETB_CHAR) 32)
#define S_PLATFORM                     ((ETB_CHAR) 8)
#define S_PRODUCT_VERSION              ((ETB_CHAR) 16)
#define S_PTIME                        ((ETB_CHAR) 8)
#define S_PUBLICATION_ID               ((ETB_CHAR) 16)
#define S_PUID                         ((ETB_CHAR) 28)
#define S_SECURITY_TOKEN               ((ETB_CHAR) 32)
#define S_SERVER_CLASS                 ((ETB_CHAR) 32)
#define S_SERVER_NAME                  ((ETB_CHAR) 32)
#define S_SERVICE                      ((ETB_CHAR) 32)
#define S_T_NAME                       ((ETB_CHAR) 8)
#define S_TOKEN                        ((ETB_CHAR) 32)
#define S_TOPIC                        ((ETB_CHAR) 96)
#define S_TXT                          ((ETB_CHAR) 40)
#define S_U_STATUS                     ((ETB_CHAR) 32)
#define S_UOW_ID                       ((ETB_CHAR) 16)
```

```
#define S_USER_ID                      ((ETB_CHAR) 32)
#define S_USRDATA                      ((ETB_CHAR) 16)
#define S_VERS                         ((ETB_CHAR) 8)
#define S_WAIT                         ((ETB_CHAR) 8)
#define S_BROKER_URL                   ((ETB_SHORT) 512)

/*--------------------------------------------------------------------*/
/* ETBCB: EntireX Broker API Control Block Definition                 */
/* (The current size is 880 bytes)                                    */
/*--------------------------------------------------------------------*/

typedef struct
{
  ETB_BYTE  api_type;                        /* v1: Type of ETBCB            */
  ETB_BYTE  api_version;                     /* v1: For compatibility        */
  ETB_BYTE  function;                        /* v1: Function                 */
  ETB_BYTE  option;                          /* v1: Option                   */
  ETB_CHAR  reserved[16];                    /* v1: Reserved for future use  */
  ETB_LONG  send_length;                     /* v1: Length of data to send   */
  ETB_LONG  receive_length;                  /* v1: Maximum receive length   */
  ETB_LONG  return_length;                   /* v1: Length of received data  */
  ETB_LONG  errtext_length;                  /* v1: Errortext buffer length  */
  ETB_CHAR  broker_id[S_BROKER_ID];          /* v1: Target broker id         */
  ETB_CHAR  server_class[S_SERVER_CLASS];    /* v1: Part of service name     */
  ETB_CHAR  server_name[S_SERVER_NAME];      /* v1: Part of service name     */
  ETB_CHAR  service[S_SERVICE];              /* v1: Part of service name     */
  ETB_CHAR  user_id[S_USER_ID];              /* v1: User id of caller        */
  ETB_CHAR  password[S_PASSWORD];            /* v1: Password of caller       */
  ETB_CHAR  token[S_TOKEN];                  /* v1: Special purposes         */
  ETB_CHAR  security_token[S_SECURITY_TOKEN];/* v1: Security purposes        */
  ETB_CHAR  conv_id[S_CONV_ID];              /* v1: Conversational/non-conv. */
  ETB_CHAR  wait[S_WAIT];                    /* v1: Blocked/non-blocked      */
  ETB_CHAR  error_code[S_ERROR_CODE];        /* v1: Error class/number       */
  ETB_CHAR  environment[S_ENVIRONMENT];      /* v1: Translation purposes     */
  ETB_LONG  adcount;                         /* v2: Attempted deliv. count   */
  ETB_CHAR  user_data[S_USRDATA];            /* v2: User data field          */
  ETB_CHAR  msg_id[S_MSGID];                 /* v2: Not used by Broker       */
  ETB_CHAR  msg_type[S_MSGTYPE];             /* v2: Not used by Broker       */
  ETB_CHAR  ptime[S_PTIME];                  /* v2: Not used by Broker       */
  ETB_CHAR  newpassword[S_PASSWORD];         /* v2: New password of caller   */
  ETB_CHAR  adapt_err[S_ADAPTERR];           /* v2: Adapter error            */
  ETB_CHAR  client_uid[S_CLIENTUID];         /* v2: Userid for security      */
  ETB_BYTE  conv_stat;                       /* v2: Conversation status      */
  ETB_BYTE  store;                           /* v2: Flag for saving data     */
  ETB_BYTE  status;                          /* v2: Not used by Broker       */
  ETB_BYTE  uowStatus;                       /* v2: UOW's status             */
  ETB_CHAR  uowTime[S_WAIT];                 /* v3: Lifetime of UOW in secs  */
  ETB_CHAR  uowID[S_UOW_ID];                 /* v3: UOW ID                   */
  ETB_CHAR  userStatus[S_U_STATUS];          /* v3: User Status              */
  ETB_BYTE  uowStatusPersist;                /* v3: UOW Status persist flag  */
  ETB_CHAR  reserved2[3];                    /* v3: Alignment                */
  ETB_CHAR  locale_string[S_LOCALE];         /* v4: Callers set_locale (ECS) */
  ETB_BYTE  data_arch;                       /* v4: For future use           */
  ETB_CHAR  forceLogon;                      /* v6: Force logon              */
  ETB_BYTE  encryptionLevel;                 /* v6: End-to-end encryption    */
  ETB_CHAR  kernelsecurity;                  /* v7: Security indicator       */
  ETB_CHAR  commitTime[S_COMMIT_TIME];       /* v7: UOW commit time          */
  ETB_CHAR  compress;                        /* v7: Compression level        */
  ETB_BYTE  reserved3[2];                    /* v7: Alignment                */
  ETB_LONG  reserved4;                       /* v7: Reserved for future use  */
  ETB_CHAR  uwStatLifeTime[S_WAIT];          /* v8: UowStatusLifetime:adder  */
  ETB_CHAR  topic[S_TOPIC];                  /* v8: Topic name               */
  ETB_CHAR  publicationID[S_PUBLICATION_ID]; /* v8: Publication ID           */
  ETB_CHAR  partner_broker_id[S_BROKER_ID];  /* v9: Partner broker id        */
  ETB_LONG  reserved_etbcb_v73_1;            /* v9: Reserved for future use  */
  ETB_LONG  reserved_etbcb_v73_2;            /* v9: Reserved for future use  */
  ETB_LONG  reserved_etbcb_v73_3;            /* v9: Reserved for future use  */
  ETB_LONG  client_id;                       /* v9: Unique client identifier */
  ETB_CHAR  reserved_etbcb_v73_4[32];        /* v9: Reserved for future use  */
  ETB_BYTE  logCommand;                      /* v9: Broker command logging   */
  ETB_BYTE  credentialsType;                 /* v9: Credentials type         */
  ETB_CHAR  reserved_etbcb_v73_5[32];        /* v9: Reserved for future use  */
  ETB_BYTE  reserved5[2];                    /* v9: Alignment                */
  ETB_LONG  varlist_offset;                  /*v10: Variable list offset     */
  ETB_LONG  long_broker_id_length;           /*v10: Length long broker id    */
} ETBCB;

/*---------------------------------------------------------------------*/
/* ATMCB: Attach Manager Control Block                                 */
/*---------------------------------------------------------------------*/

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 Servers      */
  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;

/* --------------- EntireX Broker API -------------------------------- */

#if(( __MVS__ && ( defined( __IBMC__ ) || defined( __IBMCPP__ ))) || __VSE__ )
#pragma map( broker, "BROKER" )
#endif

#if defined( __SNI )
# define broker BROKER
extern ETB_LONG BROKER( ETBCB*, ETB_CHAR*, ETB_CHAR*, ETB_CHAR* );
```

```
#elif defined( _WIN32 )
extern ETB_LONG __cdecl broker( ETBCB*, ETB_CHAR*, ETB_CHAR*, ETB_CHAR* );

#else
extern ETB_LONG broker( ETBCB*, ETB_CHAR*, ETB_CHAR*, ETB_CHAR* );
#endif

typedef ETB_LONG
#if defined( _WIN32 )
(__cdecl *PFBROKER)
#else
(*PFBROKER)
#endif
( ETBCB*, ETB_CHAR*, ETB_CHAR*, ETB_CHAR* );

#if defined( _WIN32 )
# define ETB_SHARED_LIBRARY_A "broker.dll"
# define ETB_SHARED_LIBRARY_W L"broker.dll"
# if defined( UNICODE )
#  define ETB_SHARED_LIBRARY ETB_SHARED_LIBRARY_W
# else
#  define ETB_SHARED_LIBRARY ETB_SHARED_LIBRARY_A
# endif

#elif( defined( __hpux ) && !defined( __ia64 ))
# define ETB_SHARED_LIBRARY "broker.sl"

#elif defined( __SNI )
# define ETB_SHARED_LIBRARY "BROKER2 "
# define ETB_BATCH_LOAD_MODULE "BROKER  "

#elif( __MVS__ )
# define ETB_SHARED_LIBRARY "BROKER2 "
# define ETB_BATCH_LOAD_MODULE "BROKER  "
# define ETB_CICS_LOAD_MODULE "CICSETB "

#elif( __VSE__ )
# define ETB_BATCH_LOAD_MODULE "BKIMB   "
# define ETB_CICS_LOAD_MODULE "BKIMC   "

#elif( __VMS )
# define ETB_SHARED_LIBRARY "broker.exe"

#else
# define ETB_SHARED_LIBRARY "broker.so"
#endif

#if( defined( __SNI ) || __MVS__ || __VSE__ )
# define ETB_ENTRY_POINT "BROKER"
#else
# define ETB_ENTRY_POINT "broker"
#endif
```

```
#ifdef __cplusplus
}
#endif

#endif
```

ACI Examples and Header Files

When you begin to write your first EntireX Broker ACI program, you can use the client and server examples listed below as models for your own implementation. If the examples are not available on your platform, transfer them - using FTP, for example - from a platform where they are delivered.

Depending on your platform for C, you will find the files with the examples, include files, etc. in the following locations:

Platform	Header Files / Examples	Location	Notes
z/OS	Broker ACI control block header file	See member `ETBCDEF` in the mainframe source library EXX951.SRCE.	5, 6
	Broker Command and Info Services control block header file	See member `ETBCINF` in the mainframe source library EXX951.SRCE.	5, 6
	Client example	See member `BCOC` in the mainframe source library ETB*vrs*.SRCE.	5, 6
	Server example	See member `BCOS` in the mainframe source library ETB*vrs*.SRCE.	5, 6
UNIX	Broker ACI control block header file	See `etbcdef.h` in: *include*	3, 4
	Broker Command and Info Services control block header file	See `etbcinf.h` in: *include*	3, 4
	Client example	See *convClt.c* in *examples/ACI/conversational/C/* and *nconvClt.c* in *examples/ACI/nonConversational/C/*.	3, 4
	Server example	See *convSrv.c* in *examples/ACI/conversational/C/* and *nconvSrv.c* in *examples/ACI/nonConversational/C/*.	3, 4
Windows	Broker ACI control block header file	See `etbcdef.h` in: *include*.	2
	Broker Command and Info Services control block header file	See `etbcinf.h` in: *include*.	2
	Client example	See *convClt.c* in *examples\ACI\conversational\C\* and *nconvClt.c* in *examples\ACI\nonConversational\C\*.	2
	Server example	See *convSrv.c* in *examples\ACI\conversational\C* and *nconvSrv.c* in *examples\ACI\nonConversational\C*.	2

Platform	Header Files / Examples	Location	Notes
BS2000/OSD	Broker ACI control block header file	See element `ETBCDEF.H` in the LMS library EXX951.LIB.	
	Broker Command and Info Services control block	See element `ETBCINF.H` in the LMS library EXX951.LIB.	
z/VM	Broker ACI control block header file	See member `PL1DEF` of the z/VM MACLIB EXX951.MACLIB.	10
	Broker Command and Info Services control block	See member `PL1DEF` of the z/VM MACLIB EXX951.MACLIB.	10
IBM i	Broker ACI control block header file	See member `ETBCDEF` in include source file *H_EXA*.	1, 7
	Broker Command and Info Services control block	See member `ETBCINF` in include source file *H_EXA*.	1, 7
	Sample procedure for compiling	See member `CRT_CMOD` in source file *EXASRC*.	1
	Client example	See member `BCOC` of type C in source file *EXASRC*.	1, 9
	Procedure to call client example	See the CL member `EXABCOC` in source file *EXASRC*.	1, 8
	Procedure to call Client example with Security parameters	See the CL member `EXABCOCSEC` in source file *EXASRC*.	1, 8
	Server example	See member `BCOS` of type C in source file *EXASRC*.	1, 9
	Procedure to call server example	See the CL member `EXABCOS` in source file *EXASRC*.	1, 8
	Procedure to call server example with Security parameters	See the CL member `EXABCOSSEC` in source file *EXASRC*.	1, 8

Notes:

1. See *Installing EntireX under IBM i*.
2. See *Post-installation Steps under Windows*.
3. For information on *`exxdir`*, see *Shell Environment Settings* under *Post-installation Steps under UNIX*.
4. See *Post-installation Steps under UNIX*.
5. See *Installing EntireX under z/OS* in the z/OS installation documentation.
6. For information on *`vrs`*, see *Contents of Mainframe Installation Medium* in the z/OS installation documentation.
7. Rename file H_EXA to H before use it.
8. By default, these CL procedures call the C-type of the client and server programs - i.e, `BCOC` and `BCOS`. Modify the procedures to adjust the Broker ID, Broker Version and Security parameters.

Compile the sources and bind the created modules to executable *PGM programs. For compilation, use the procedure CRT_CMOD. For binding, use the procedure EXABNDPGM. All sample programs include the ACI Broker control block definitions ETBCDEF during compilation.

9. See also *Verifying the Installation of the Broker Stubs* in the IBM i installation documentation.
10. See *Installing Broker Stubs under z/VM*.

Creating a C User Application under IBM i

On the IBM i system, the broker stub is implemented as an object of type *SRVPGM (Service Program). This object type has the advantage that its program code can be shared by several programs. It exists as an object on its own and can therefore be easily replaced without rebinding the user's application, when a newer version becomes available.

The service program EXA supplied by Software AG contains all the functions necessary for controlling and communicating with the remote broker. To create an executable Broker application on IBM i, you need to develop, in any ILE-enabled programming language, at least one main module to which the EXA service program is bound.

- For compilation, use the command CRTCMOD with the options:

```
...DEFINE (CE_TAS400 TCP_IP '_MULTI_THREADED')
...SYSIFCOPT (*NOIFSIO)...
```

- For binding, use the command CRTPGM with the option:

```
...BNDSRVPGM(*LIBL/EXA) ...
```

Example:

The following steps show how to create a server application using the program BCOS. See ***ACI Examples and Header Files***.

Step 1: Set the Environment

The library EXX must be located in the *LIBL list.

To set the library list, you can use the command:

```
CHGCURLIB CURLIB(EXX)
```

Step 2: Compile the User Program

To compile BCOS, use the command CRTCMOD with options similar to the following:

```
MODULE(BCOS) SRCFILE(*CURLIB/EXASRC) OUTPUT(*PRINT) DEFINE(CE_TAS400
TCP_IP '_MULTI_THREADED') SYSIFCOPT(*NOIFSIO)
```

Or, use the sample procedure CRT_CMOD

If the program has been successfully compiled, the module BCOS will be created.

Step 3: Bind EXA to the User Program

To produce an executable program, bind the user program BCOS to the service program EXA supplied by Software AG. Use the command CRTPGM similar to the following:

```
CRTPGM PGM(EXX/BCOS) MODULE(*PGM) ENTMOD(*PGM)
BNDSRVPGM(EXX/EXA) BNDDIR(*NONE) OPTION(*GEN *WARN *DUPVAR)
DETAIL(*EXTENDED)
```

Or, use the sample procedure EXABNDPGM.

If the programs have been bound successfully, the object BCOS with type *PGM will be created.

2 Writing Applications: Publish and Subscribe

This chapter describes how to implement and program publish-and-subscribe applications - employing durable subscription techniques - with EntireX Broker. Publish-and-subscribe communication is used if data is to be published in order to make it available to one or more subscribers. This communication model is implemented as an independent subsystem in EntireX Broker, that is, it can be activated by setting attributes or left inactive.

For ease of use, we recommend you use the aids and techniques below in the order given.

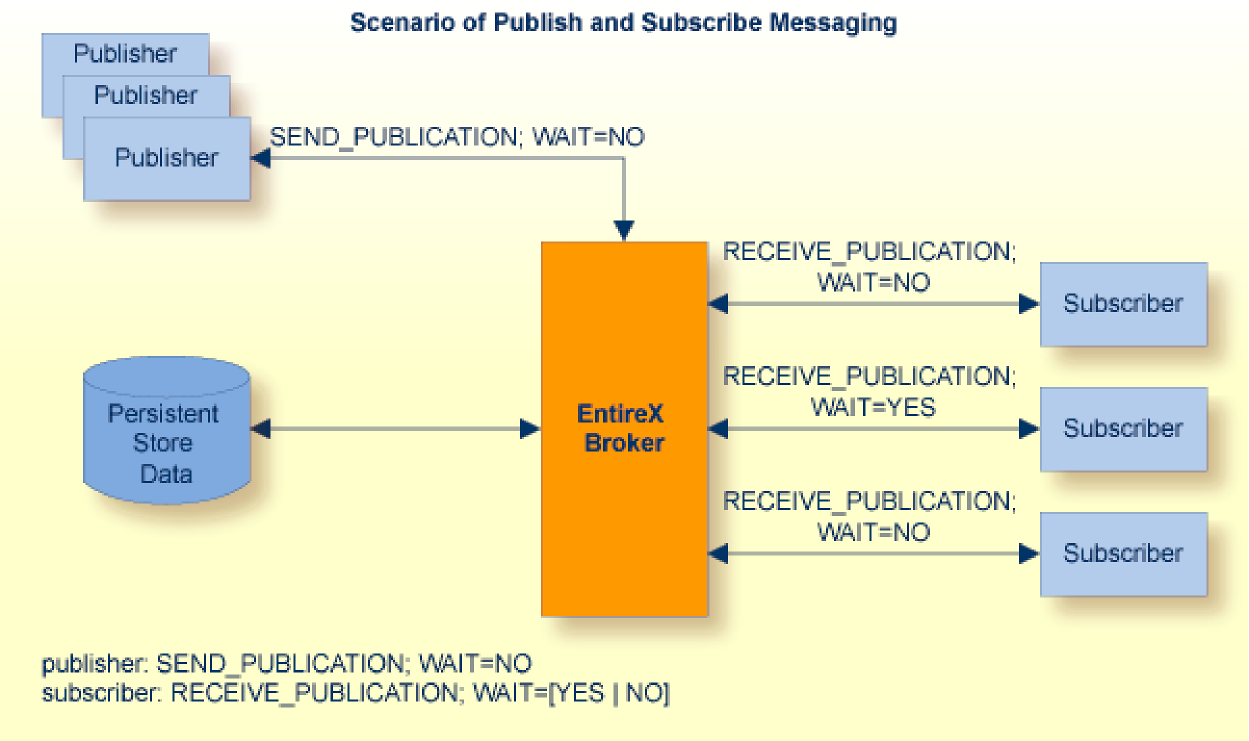
See also *Concepts of Persistent Messaging* in the general administration documentation.

Overview of Communication Models

There are two communication models in EntireX Broker: publish and subscribe and client and server.

- **Publish and Subscribe**
  This communication model is used if data is to be published to multiple recipients. It is an alternative to client and server and is implemented as an independent subsystem in EntireX Broker.
- **Client and Server**
  This communication model is based on a logical connection between exactly two partners: a client and a server. It covers these communication requirements conversationally and non-conversationally, and synchronously and asynchronously. See *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

Basic Concepts of Publish and Subscribe


Scenario of Publish and Subscribe Messaging
Publisher
Publisher
Publisher
SEND_PUBLICATION; WAIT=NO
EntireX Broker
RECEIVE_PUBLICATION; WAIT=NO
Subscriber
RECEIVE_PUBLICATION; WAIT=YES
Subscriber
RECEIVE_PUBLICATION; WAIT=NO
Subscriber
Persistent Store Data
publisher: SEND_PUBLICATION; WAIT=NO
subscriber: RECEIVE_PUBLICATION; WAIT=[YES | NO]

Topic

A topic is a logical grouping of publications relating to one subject area, which is defined in the *Broker Attributes* in the administration documentation. Topics reflect subject areas, for example current news, stock quotations, weather, online chat, sales systems. Data can be published to a topic only if there are current subscribers to this topic.

Note: For EntireX Broker, the term “topic” is analogous to the term “service”. Topic represents the grouping of related information flows for the publish-and-subscribe communication model, as service does for the client-and-server model.

Publication

A publication is a message or set of messages that are created atomically by one publisher and are available to all current subscribers to the topic. Messages for publication are queued to the topic on a first-in, first-out basis.

Each publication is assigned a unique `PUBLICATION-ID` by EntireX Broker when the publication is created. The `PUBLICATION-ID` is returned to the publisher on the first `SEND_PUBLICATION` command issued when creating a new publication. The `PUBLICATION-ID` is also returned to the subscriber on the first `RECEIVE_PUBLICATION` command used to receive each new publication. Publisher and subscriber must include the `PUBLICATION-ID` for all subsequent commands relating to the same publication.

Subscription

A subscription identifies a user's intention to receive publications for a specified topic. An active subscription requires the user to have issued a `SUBSCRIBE` command without issuing a subsequent `UNSUBSCRIBE` command. Only publications created after the time of subscription can be read by the subscriber. Conversely, publications created after the `UNSUBSCRIBE` command, or after the subscription has expired, cannot be received by the subscriber, even if the subscription is renewed at a later time. The time period of the subscription determines which publications can be delivered to the subscriber. See *Durability of Subscriptions* under *Basic Concepts of Publish and Subscribe* under *Writing Applications: Publish and Subscribe* in the ACI Programming documentation.

Publisher

A publisher is a user participating in publish and subscribe that creates publications for one or more topics. It is possible for a publisher to create publications only if there is currently at least one subscription to the topic. This prevents superfluous data from being assigned to the topic.

Subscriber

A subscriber is a user participating in publish and subscribe that can read publications from one or more topics.

Durability of Subscriptions

The behavior of a subscription when the subscriber logs off or broker shuts down is determined by an option specified in the original subscription command.

- **Durable Subscription**

  EntireX Broker enables publish-and-subscribe applications to execute with durable subscriptions by maintaining the user's subscription status on disk. This ensures that - in the case of a system failure - subscriber information will automatically be recovered, allowing applications to be restarted without any loss of data.

If `DURABLE` is specified within the `SUBSCRIBE` command, users need only subscribe once to a topic. The subscription is retained after the user issues a `LOGOFF` command or if the subscriber has timed out. Similarly, the subscription remains if the broker is restarted. All publications necessary to satisfy subscription requirements are also retained. If a subscriber no longer wishes to subscribe to a topic, the subscriber must issue an `UNSUBSCRIBE` command; otherwise the subscription remains valid until the subscription expiration time has passed. Durable subscription requires the administrator to configure the persistent store. See *Concepts of Persistent Messaging* in the general administration documentation. In addition, the topic must be specified as durable in the *Broker Attributes* in the administration documentation.

Durable subscription comprises:

- a list of subscribers and topics to which subscribers have durably subscribed;
- information about the last publication received.

A user has to subscribe only once to a topic. The persistent status remains after the broker is restarted. A subscriber signals its intention to receive publications by issuing a `SUBSCRIBE` command and specifying the topic of interest. If the administrator has specified this topic in the broker's attribute file with a characteristic of `ALLOW-DURABLE`, users will be able to subscribe durably to the topic.

- **Non-durable Subscription**

  Publish-and-subscribe applications can also employ non-durable subscription techniques, if desired. Publications (messages sent from publishers to subscribers) can be either durable or non-durable.

  If durable is not specified in the subscribe command, the subscription is valid only until one of the following events occurs, after which subscription is terminated and publication can no longer be retrieved:

  - the user issues a `LOGOFF` command;
  - Broker is restarted;
  - the subscriber non-activity time value has passed;

    or

  - the subscription expiration time has passed.

The time at which the `SUBSCRIBE` command is issued is significant to the user's subscription. Only publications created after this point in time can be read by the subscriber. Conversely, publications created after either the time at which the `UNSUBSCRIBE` command was issued or the subscription has expired cannot be received by the subscriber, even if the subscription is renewed at a later time. The time period of the subscription determines which publications are delivered.

It is possible for a publisher to create publications only if there is currently at least one subscription to the topic. This feature prevents superfluous data from being assigned to the topic.

Subscription Expiration

A topic is specified in the broker's attribute file with a characteristic of SUBSCRIPTION-EXPIRATION time. This is the time period for which the user's subscription remains in effect. After the time period has elapsed, the user's subscription is terminated and the subscription is removed by the broker.

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. Using publish and subscribe requires API-VERSION 8 or higher. 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 broker function is required in order to use the publish-and-subscribe programming model in your application. We recommend that the application issue a LOGOFF function call for the following reasons:

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

Logon 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

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 with Durable Subscription

> **Caution:** USER and TOKEN must be specified by all publisher and subscriber applications where publication and subscription data is held in the persistent store.

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. This specification is also necessary to enable effective use of publish and subscribe, including recovery from system failures.

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

This section describes the basic functionality of the API. There are five distinct functions in the Broker ACI which are relevant to publish and subscribe:

- CONTROL_PUBLICATION

  The function CONTROL_PUBLICATION is used by both the publisher and the subscriber. The publisher uses CONTROL_PUBLICATION,OPTION=COMMIT to commit the publication it is creating; the subscriber uses CONTROL_PUBLICATION,OPTION=COMMIT to acknowledge the receipt of the publication it is receiving.

- RECEIVE_PUBLICATION

  The function RECEIVE_PUBLICATION is used by the subscriber to receive all or part of a publication. The field PUBLICATION-ID defines the behavior of this function. RECEIVE_PUBLICATION,PUBLICATION-ID=NEW signals the subscriber's readiness to obtain the next available new publication, whereas the value PUBLICATION-ID=*nnn* specifies that the next message within an existing publication is being requested. After all messages have been received, the publication is acknowledged, using the function CONTROL_PUBLICATION,OPTION=COMMIT.

- SEND_PUBLICATION

  The function SEND_PUBLICATION is used by the publisher to produce a publication. The field PUBLICATION-ID defines the behavior of this function. The publisher uses SEND_PUBLICATION,PUBLICATION-ID=NEW to create a new publication. The value PUBLICATION-ID=*nnn* indicates that a subsequent message within the same publication is being sent, which can be necessary when creating large publications. A publication is completed with the function SEND_PUBLICATION,OPTION=COMMIT or with the function call CONTROL_PUBLICATION, using the option COMMIT.

- SUBSCRIBE

  The function SUBSCRIBE registers a user with the broker as a subscription for a certain topic. Specifying SUBSCRIBE,OPTION=DURABLE determines that the subscription is to be durable. Otherwise the subscription is non-durable.

- UNSUBSCRIBE

  The function UNSUBSCRIBE covers the opposite functionality: a subscription is cancelled or dissolved.

The participants in publish-and-subscribe are identified by ACI fields USER-ID and TOKEN.

ACI Syntax

<table>
<tr><th>Function</th><th>Fields in EntireX Broker Control Block</th></tr>
<tr><td>CONTROL_PUBLICATION</td><td>API = 8
, BROKER-ID = BROKER-ID
, USER-ID = user_id
, TOKEN = token
, OPTION = { BACKOUT |
CANCEL |
COMMIT |
LAST |
QUERY |
SETUSTATUS }
[, PUBLICATION-ID = pub_id]
[, USTATUS = user_status]</td></tr>
<tr><td>RECEIVE_PUBLICATION</td><td>API = 8
, BROKER-ID = BROKER-ID
, USER-ID = user_id
, TOKEN = token
, WAIT = NO | YES | wait_value
, PUBLICATION-ID = pub_id | NEW | OLD | ANY
, TOPIC = topic_name</td></tr>
<tr><td>SEND_PUBLICATION</td><td>API = 8
, BROKER-ID = BROKER-ID
, USER-ID = user_id
, TOKEN = token
[, OPTION = COMMIT]
, PUBLICATION-ID = pub_id | NEW
, TOPIC = topic_name
[, USTATUS = user_status]</td></tr>
<tr><td>SUBSCRIBE</td><td>API = 8
, BROKER-ID = BROKER-ID
, USER-ID = user_id
, TOKEN = token
, TOPIC = topic_name
[, OPTION = DURABLE]</td></tr>
<tr><td>UNSUBSCRIBE</td><td>API = 8
, BROKER-ID = BROKER-ID
, USER-ID = user_id
, TOKEN = token
, TOPIC = topic_name</td></tr>
</table>

Key ACI Field Names

The following table lists key ACI field names used to implement applications that use the publish-and-subscribe communication model. The other fields are available to identify partner programs, specify buffer lengths, convey error codes, etc.

See *Broker ACI Fields* in the ACI Programming documentation for all fields.

ACI Field Name	Description
`FUNCTION`	Function code for one of the verbs (see table below).
`OPTION`	Indication of specific broker behavior, depending on the function.
`PUBLICATION-ID`	Identifier to obtain and specify the publication. Indicates a specific publication. The publication ID value is an internally generated identifier (containing alphanumeric characters) for the publication. We recommend that application programmers make no assumptions about the content, layout or meaning of any part of the `PUBLICATION-ID` field.
`TOPIC`	Identifies the name of the publication's topic.
`WAIT`	Value to specify blocking or non-blocking command.

Key Verbs for FUNCTION Field

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

See *Broker ACI Functions* in the EntireX Broker ACI Programming documentation for all functions.

Verb	Description
`CONTROL_PUBLICATION`	Publisher uses this to commit and subscriber uses this to acknowledge publications.
`RECEIVE_PUBLICATION`	Retrieves publication from the broker.
`SEND_PUBLICATION`	Sends publication to the broker.
`SUBSCRIBE`	Informs the broker of the existence of a subscriber to a topic.
`UNSUBSCRIBE`	Informs the broker that the subscriber wishes to unsubscribe.

Implementation of Publisher and Subscriber Components

Single Message Publication

This example illustrates a publisher creating single-message publications that are retrieved by one or more subscriber applications. The publisher and subscriber operate asynchronously of each other. There is no reply from the subscriber in this communication model.

This example, which uses durable subscription, shows the typical structure of a subscriber application that has previously subscribed to a topic and is now retrieving the publications issued to that topic. Subscription occurs either during one-time processing provided by the subscriber application, or it is performed explicitly by an administrator. See *Broker Command and Information Services*.

The subscriber performs RECEIVE_PUBLICATION commands in a loop specifying WAIT=YES, which makes it possible to process publications as they occur. If none are received during the specified wait period, the server executes another iteration of the loop and repeats the wait until a publication is received.

The RECEIVE_PUBLICATION command specifies PUBLICATION-ID=NEW to receive all new publications arriving from the publisher. This example assumes single-message publications which do not require acknowledgment of receipt since AUTO-COMMIT-FOR-SUBSCRIBER=YES was specified in the topic-specific attributes for the topic NYSE in this case.

```
LOGON USER-ID=SB1,TOKEN=TKSB1
Repeat
    ↩
RECEIVE_PUBLICATION,PUBLICATION-ID=NEW,WAIT=YES,TOPIC=NYSE,USER-ID=SB1,TOKEN=TKSB1
        If (Error-Class = 0 and Error-Number = 0)
        /* something received: process request*/

        End-if /* otherwise nothing received */
End-repeat
LOGOFF USER-ID=SB1,TOKEN=TKSB1
```

A publisher issues a SEND_PUBLICATION command to send publications containing a single message to a topic. The publisher's SEND_PUBLICATION commands are performed with WAIT=NO, and PUBLICATION-ID=NEW is assigned each time.

```
SEND_PUBLICATION,PUBLICATION-ID=NEW,WAIT=NO,OPTION=COMMIT,TOPIC=NYSE,USER-ID=PB1,TOKEN=TKPB1
```

Multiple Message Publication

This example, which uses durable subscription, shows a publisher creating multiple-message publications that are retrieved by one or more subscriber applications. The publisher and subscriber operate asynchronously of each other; there is no reply from the subscriber in this communication model. In this example, one or more publishers in a stock exchange system send current stock exchange quotations. The subscriber accesses the system at irregular intervals and receives all publications currently available.

This example illustrates the typical structure of a subscriber application that has previously subscribed to a topic and is now retrieving all available publications for a specified topic. Subscription has already occurred either during one-time processing within the subscriber application, or it is performed explicitly by an administrator. See *Command-line Utilities* under *Broker Command and Information Services*.

The subscriber performs `RECEIVE_PUBLICATION` commands in the outer loop, specifying `PUBLICATION-ID=NEW` in order to receive the first available publication. The inner loop allows remaining messages within the same publication to be retrieved, after which `CONTROL_PUBLICATION` acknowledges receipt of the publication. The outer loop is then repeated to obtain the next available publication in conjunction with the inner loop until all available publications are processed.

The `RECEIVE_PUBLICATION` command specifies `PUBLICATION-ID=NEW` to receive all new publications. In this case, the subscriber explicitly acknowledges receipt of the publication, using the `CONTROL_PUBLICATION` function, since it is assumed `AUTO-COMMIT-FOR-SUBSCRIBER` was not specified in the topic-specific attributes for the topic NYSE in this case.

```
LOGON USER-ID=SB1,TK=TKSB1
While publications available
  RECEIVE_PUBLICATION PUBLICATION-ID=NEW,TOPIC=NYSE,WAIT=YES,USER-ID=SB1,TOKEN=TKSB1
  While data on publication
    RECEIVE_PUBLICATION ↩
PUBLICATION-ID=publication-id,TOPIC=NYSE,WAIT=NO,USER-ID=SB1,TOKEN=TKSB1
  End-while
  CONTROL_PUBLICATION OPTION=COMMIT,PUBLICATION-ID=publication-id,TOPIC=NYSE
End-while
LOGOFF USER-ID=SB1,TOKEN=TKSB1
```

A publisher issues a `SEND_PUBLICATION` command to send a publication containing multiple messages. The publisher's `SEND_PUBLICATION` command is performed with `WAIT=NO` and `PUBLICATION-ID=NEW`. Remaining messages belonging to this publication are sent to the broker by specifying the generated `PUBLICATION-ID` within each subsequent `SEND_PUBLICATION` command. These messages are committed by issuing the `CONTROL_PUBLICATION` command, which also specifies the generated `PUBLICATION-ID`.

```
LOGON USER-ID=PB1,TK=TKPB1
SEND_PUBLICATION PUBLICATION-ID=NEW,TOP=NYSE,USER-ID=PB1,TOKEN=TKPB1
While data
  SEND_PUBLICATION PUBLICATION-ID=publication-id,USER-ID=PB1,TOKEN=TKPB1
End-while
CONTROL_PUBLICATION ↩
OPTION=COMMIT,PUBLICATION-ID=publication-id,USER-ID=PB1,TOKEN=TKPB1
LOGOFF USER-ID=PB1,TOKEN=TKPB1
```

Blocked and Non-blocked Broker Calls

In the publish-and-subscribe communication model, the term “blocked call” refers only to the broker `RECEIVE_PUBLICATION` command used by subscriber applications. The `SEND_PUBLICATION` command is always “non-blocking”, such that `WAIT=NO` must be specified. A publisher application sends a publication via EntireX Broker for a specified topic without waiting for any subscribers to receive the publication.

A subscriber application component can use the control block field `WAIT` in the following ways to determine whether broker will automatically generate a `WAIT` in order for the command to be either received or satisfied by the partner application:

Non-blocked Command: WAIT=NO

`RECEIVE_PUBLICATION` allows a subscriber application to request a publication for a specified topic. If there are no publications currently available, an ACI response code is returned, indicating that no publications are currently available for the designated topic. See *Error Messages and Codes*. Similarly, a response code also indicates that there are no further messages to be received within the same publication, where `PUBLICATION-ID=`*nnn* has been specified to retrieve continuation segments of the same publication. This technique is used by subscriber applications only.

Example: Subscriber

The subscriber application component requests the next new publication, which is returned if available. If there is no publication available, the subscriber receives a return code immediately, indicating no publications are available at this time. There is no waiting, and the application performs this command periodically under control of the application logic, as shown here:

```
RECEIVE-PUBLICATION,PUBLICATION-ID=NEW,WAIT=NO,TOPIC=NYSE,USER-ID=SB1,TOKEN=TKSB1
... application code to process publication ....
```

Blocked Command: WAIT=YES or WAIT=n

Allows a subscriber application to solicit a publication to be returned for the specified topic. The calling application is automatically placed in a `WAIT` state until there is a publication available for the specified topic. If no publication is available during the specified waiting time, an ACI response code is returned to the application, indicating that no publications are currently available for the designated topic. See *Error Messages and Codes*. Similarly, a response code also indicates that there are no further messages to be received within the same publication if `PUBLICATION-ID=`*nnn* has been specified in order to retrieve continuation segments of the same publication. This technique is used by subscriber applications only.

Example: Subscriber

The subscriber application component requests the next new publication, which is returned if available. If there is no publication available, the subscriber enters a `WAIT` state for the specified (or default) time period, during which it is eligible to receive any new publications that arrive in this time. At the end of the specified (or default) time period, the subscriber receives a return code if no publications were available. The following example shows this process being repeated indefinitely within a loop:

```
Repeat
   RECEIVE-PUBLICATION,PUBLICATION-ID=NEW,WAIT=YES,TOPIC=NYSE,USER-ID=SB1,TOKEN=TKSB1
    ... application code to process publication ....
End-repeat
```

Timeout Parameters

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 `PUBLISHER-NONACT` and `SUBSCRIBER-NONACT` attributes are non-activity timeout parameters which can be specified independently of each other to control the timeout behavior of publisher and subscriber application components. If an application component issues no commands to the broker for the specified time period, the broker logs the user off automatically, cleaning up related in-memory resources. See *`LOGON` and `LOGOFF`* under *Writing Applications: Publish and Subscribe* in the ACI Programming documentation. If the subscriber did not issue a durable `ALLOW-DURABLE` command, the user's subscription will also be removed after this time.
- The `SUBSCRIPTION-EXPIRATION` attribute determines the lifetime of a user's durable subscription. Durable subscriptions are retained by the broker until either the subscriber issues an `UNSUBSCRIBE` command or the subscription lifetime has expired.
- The `PUBLICATION-LIFETIME` attribute determines how long publications are retained by the broker until they are either received by all subscribers or the publication lifetime has expired.
- The `WAIT` field in the ACI control block is significant only to the subscriber application component. The program is placed into a `WAIT` state for a specified time when issuing the `RECEIVE_PUBLICATION` command, allowing 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 making a blocked command. A non-blocked command is executed if `WAIT=NO` is specified. See *Blocked and Non-blocked Broker Calls* under *Writing Applications: Publish and Subscribe* in the ACI Programming documentation. The `SEND_PUBLICATION` command is always issued with `WAIT=NO`.

Types of Non-activity Time

There is interplay between the non-activity times specified in the attribute file for the attributes

- `PUBLISHER-NONACT`
- `SUBSCRIBER-NONACT`
- `CLIENT-NONACT`
- `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. This fact must be considered where an application component implements both publish and subscribe and client and server.

Recommendations

The following recommendations apply to developing publish-and-subscribe applications:

- If the subscriber issues blocked `RECEIVE_PUBLICATION` commands, make the `WAIT` time adjustable. The `WAIT` value can be read as a startup parameter from the user-written INI or CFG file, or any other parameter data set or set of environment variables, depending on the platform.
- When using non-durable subscriptions, ensure the specified `SUBSCRIBER-NONACT` time is not exceeded by the subscriber between issuing commands to the broker; otherwise the user will be automatically logged off, and the user's subscription will be removed during a period of inactivity. See *Durability of Subscriptions* under *Basic Concepts of Publish and Subscribe* under *Writing Applications: Publish and Subscribe* in the ACI Programming documentation

  Note: When blocking `RECEIVE_PUBLICATION` commands, the `SUBSCRIBER-NONACT` value is overridden by the `WAIT` time (if this is greater).

- If there are no available messages for the duration of a blocked `RECEIVE_PUBLICATION` command, response code 00740074 is returned to the subscriber. The subscriber can reissue the `RECEIVE_PUBLICATION` command repeatedly until the next publication becomes available. See also ***Blocked Command: `WAIT=YES` or `WAIT=n`***.
- If there are no available messages when issuing a non-blocked `RECEIVE_PUBLICATION`, the command returns response code 00030488. See also ***Non-blocked Command: `WAIT=NO`***.
- If there are no further messages available when issuing a `RECEIVE_PUBLICATION`, the command returns response code 00740480.
- Ensure that the `PUBLISHER-NONACT` time is not exceeded by the publisher between issues of `SEND_PUBLICATION` commands; otherwise the user will be automatically logged off, and any unfinished or uncommitted publications will be lost.

Configuration Prerequisites for Durable Subscriptions

A subscription can be durable or non-durable. See *Durability of Subscriptions* under *Basic Concepts of Publish and Subscribe* under *Writing Applications: Publish and Subscribe* in the ACI Programming documentation. Durable subscriptions require additional configuration steps. Since subscriber information for durable subscriptions must also be present after a broker is restarted, a persistent store is required (`PSTORE`). See also *Concepts of Persistent Messaging* in the general administration documentation. This allows Adabas (all platforms), file system (UNIX and Windows) and DIV (z/OS) to be utilized for storing both publication information and, optionally, subscription information.

If you use the persistent store for subscriber information under Adabas, see *Configuring and Operating the Adabas Persistent Store* in the platform-specific administration documentation. If you are using persitent store type DIV or the local file system, no additional PSTORE configuration is required. See also *Broker-specific Attributes* under *Broker Attributes* in the administration documentation for other related parameters.

```
NUM-TOPIC-TOTAL        = 4
NUM-SUBSCRIBER-TOTAL   = 8
SUBSCRIBER-STORE       =PSTORE
```

Note: The topic attribute definitions must specify `ALLOW DURABLE=YES`. Otherwise durable subscription requests are rejected.

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* in the general administration documentation. 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 Progamming 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 Internationalization

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

This section covers the following topics:

General Information

The broker stub does not convert your application data before it is sent to the broker. The application's data is shipped as given.

For the internationalization approaches ICU conversion and SAGTRPC user exit, valid locale strings are required for conversion to behave correctly.

Providing Locale Strings

Under the Windows operating system:

- The broker stub assumes by default that 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 and communicating with a broker version 7.2.*n* or above, a codepage identifier of this Windows ANSI codepage is also automatically transferred as part of the locale string to tell the broker how the data is encoded.
- If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation.

Under all other operating systems:

- The broker stub does not automatically send a codepage identifier to the broker as part of the locale string.
- The broker stub assumes the broker's locale string defaults match. If they do not match, provide the codepage explicitly. See *Broker's Locale String Defaults* under *Locale String Mapping* in the internationalization documentation.

With the ACI control block field `LOCALE-STRING`:

- You can override or provide a codepage in the locale string sent to the broker. If a codepage is provided, it must follow the rules described under *Locale String Mapping* in the internationalization documentation.

- You can force a locale string to be sent if communicating with broker version 7.1.x and below. Under Windows you can use the abstract codepage name. See *Using the Abstract Codepage Name LOCAL* under *Locale String Mapping* in the internationalization documentation.
- API version 4 or above is required to override the locale string.

The encoding in which your application gives the data to the broker stub and the locale string

- must always match, i.e. the codepage derived after the broker's built-in locale string mapping process must be the same as the encoding of the data provided. See *Broker's Built-in Locale String Mapping* under *Locale String Mapping* in the internationalization documentation.
- must be a codepage supported by the broker, depending on the internationalization approach;

otherwise, unpredictable results will occur.

Example for Assembler

```
MVC   S$LOCALE,=C'ECS037'                 MOVE CP
....
```

Examples for C

1. Using a specific codepage

   ```
   /* prepare the locale-string with a codepage */
   memset (pETBCB->locale_string,' ',sizeof(pETBCB->locale_string));
   strncpy(pETBCB->locale_string,"ECS0819",sizeof(pETBCB->locale_string));
   ....
   ```

2. Using the platform's default codepage (Windows only)

   ```
   /* prepare the locale-string with a codepage */
   memset (pETBCB->locale_string,' ',sizeof(pETBCB->locale_string));
   strncpy(pETBCB->locale_string,
   ETB_CODEPAGE_USE_PLATFORM_DEFAULT,sizeof(pETBCB->locale_string));
   ....
   ```

Example for COBOL

```
MOVE 'ECS037' TO LOCALE-STRING.
....
```

Examples for Natural

```
MOVE 'ECS037' TO #SDPA-API.#LOCALE_STRING.
....
```

Using the ENVIRONMENT Field with the Translation User Exit

Using the internationalization approach *translation user exit*, an ACI programmer can provide additional information to their translation exit through the `ENVIRONMENT` field, allowing flexible translation behavior in accordance with application requirements. The field cannot be used for any other internationalization approaches and must be empty if a method other than translation user exit is used. See *Translation User Exit* under *Introduction to Internationalization*.

Example

Assume a broker service or topic has a user-written translation routine called ABCTRAN, which is capable of performing several types of data conversion, for example EBCDIC-ASCII translation, byte swapping, and mixed data types. The user translation routine may need to know the data formats used by both partners. The `ENVIRONMENT` field can be used to pass this information from the application to the translation routine in Broker kernel.

Technique

```
MOVE 'MYCODEPAGE' TO #ETBCB.#ENVIRONMENT
...
CALL 'BROKER' #ETBAPI #SEND-BUFF #RECV-BUFF #ERR-TXT
```

Using Send and Receive Buffers

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` and `RECEIVE_PUBLICATION` functions 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

Conversion and translation 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 internationalization approach. See *Internationalization with EntireX*. Note the following:

For translation and 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 or RECEIVE_PUBLICATION 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 or RECEIVE_PUBLICATION 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. **Note:** Under z/OS with Adabas version 8, the value for NET is the same as for TCP and SSL.
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 *Administration of Broker Stubs under z/OS* in the z/OS administration documentation.

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 *Administration of Broker Stubs under z/OS* in the z/OS administration documentation.

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

Transport Methods

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)** [6]
z/OS [2]	Batch, TSO, IMS (BMP)	BROKER	x	x	x	
	Com-plete	COMETB	x	[3]	x	
	CICS	CICSETB	x	[3]	x	
	IMS (MPP)	MPPETB	x	x	x	
	IDMS/DC [4]	IDMS	x	[3]		
	Natural	NATETB23	x	x	x	
	UNIX System Services	*Java ACI* in the Developer's Kit documentation	x	x		x
UNIX		broker.so	x	x		
		Java ACI in the Developer's Kit documentation	x	x		x
Windows		broker.dll [5]	x	x		
		Java ACI in the Developer's Kit documentation	x	x		x
BS2000/OSD	Batch, Dialog (formerly TIAM)	BROKER	x	x	x	
z/VM		BKIMBCMS	x		x	
IBM i		EXA	x			
OpenVMS		BROKER	x	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 you can use IBM's Application Transparent Transport Layer Security (AT-TLS) as an alternative to direct SSL support inside the broker stub. Refer to the IBM documentation for more information.
3. Use AT-TLS. See Note 2.
4. Tracing and transport timeout are not supported in this environment.

5. Stub broker32.dll is supported for reasons of backward compatibility. The functionality is identical to broker.dll.
6. Via Broker HTTP(S) Agent; see *Settting up and Administering the Broker HTTP(S) Agent* in the UNIX and Windows administration documentation.

See also:

- *Setting 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 *Setting 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 and TLS

Application programs using Secure Sockets Layer (SSL) or Transport Layer Security (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 or TLS transport. See `SETSSLPARMS` and *Running Broker with SSL or TLS Transport* in the platform-specific administration documentation.

Example: An application communicating through SSL or TLS would specify on each command the Broker ID:

```
MYPC:1958:SSL
```

where the host on which the Broker kernel executes is known to SSL or TLS as MYPC and is listening on port 1958.

Transport Examples

- **For programming language C under Windows:**

  ```
  strcpy( pSBuf, "TRUST_STORE=c:\\certs\\CaCert.pem&VERIFY_SERVER=N");
  EtbCb.send_length = strlen(pSBuf);
  EtbCb.errtext_length = 40;
  EtbCb.function = FCT_SETSSLPARMS
  rc = broker (etbcb, pSBuf, (char *) 0, pEBuf);
  ```

- **For programming language Natural under z/OS:**

  ```
  MOVE 'TRUST_STORE=UID/KEYRING' TO #SSL-BUFF
  MOVE 80 TO #ETBCB.#SEND-LENGTH MOVE 40 TO #ETBCB.#ERRTEXT-LENGTH
  MOVE #FCT-SSLP TO #ETBCB.#FUNCTION
  MOVE 'IBMHOST:1958:SSL' TO #ETBCB.#BROKER-ID
  ...
  CALL 'BROKER' #ETBAPI #SSL-BUFF #RECV-BUFF #ERR-TXT
  ```

See table above for how SSL or TLS is supported depending on broker stub and platform.

For information on Secure Sockets Layer, see *SSL or TLS and Certificates with EntireX*.

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 or RECEIVE_PUBLICATION 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* in the ACI Programming documentation.

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.

IAF Authentication

With ACI field CREDENTIALS-TYPE you can specify that the token specified in the IAF token field is used to authenticate the user. If this field is left blank, user ID and password will be used as before.

3 Writing Applications: Client and Server

This chapter describes how to implement and program client and server applications with EntireX Broker.

See also *Writing Applications: Attach Server* in the ACI Programming documentation and *Writing Applications: Units of Work* in the ACI Programming documentation.

Overview of Communication Models

There are two communication models in EntireX Broker: client and server and publish and subscribe.

- **Client and Server**
  This 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.
- **Publish and Subscribe**
  This communication model is used if data is to be published to multiple recipients. It is an alternative to client and server and is implemented as an independent subsystem in EntireX Broker. See *Writing Applications: Publish and Subscribe* in the ACI Programming documentation.

Basic Concepts of Client and Server

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* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

- **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`. Using publish and subscribe requires `API-VERSION` 8 or higher. 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

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

Caution: USER and TOKEN must be specified by all publisher and subscriber applications where publication and subscription data is held in the persistent store.

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. This specification is also necessary to enable effective use of publish and subscribe, including recovery from system failures.

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

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* under *Writing Applications: Units of Work* in the ACI Programming documentation.

- `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

<table>
<tr><th>Function</th><th>Fields in EntireX Broker Control Block</th></tr>
<tr><td>DEREGISTER</td><td>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]</td></tr>
<tr><td>EOC</td><td>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]</td></tr>
<tr><td>RECEIVE</td><td>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 | *</td></tr>
<tr><td>REGISTER</td><td>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]</td></tr>
<tr><td>SEND</td><td>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</td></tr>
</table>

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* in the ACI Programming documentation 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* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

Key Verbs for FUNCTION Field

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

See *Broker ACI Functions* in the EntireX Broker ACI Programming documentation 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
- Examples: Programming Language Natural

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* in the ACI Programming documentation).
- 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* in the ACI Programming documentation).

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* in the ACI Programming documentation). 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.

Examples: Programming Language Natural

- **Blocked Broker Calls**
  - *Example 1: Single Request without Reply* under *Examples for EntireX Broker Tutorial*
  - *Example 2: Single Request with Reply* under *Examples for EntireX Broker Tutorial*
- **Non-blocked Broker Calls**
  - *Example 3: Long Running Service - Non-blocked Client* under *Examples for EntireX Broker Tutorial*
  - *Example 14: Single Requests without Reply - A Polling Server* under *Examples for EntireX Broker Tutorial*
  - *Example 15: Single Requests with Reply - A Polling Server* under *Examples for EntireX Broker Tutorial*

Note: See *Examples for EntireX Broker Tutorial*.

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* under *Error Messages and Codes* is received. See *Error Handling* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

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* under *Error Messages and Codes*; the error number specifies the actual circumstances.

Examples: Programming Language Natural

- Non-conversational communication
  - *Example 1: Single Request without Reply* under *Examples for EntireX Broker Tutorial*
  - *Example 2: Single Request with Reply* under *Examples for EntireX Broker Tutorial*
- Conversational communication
  - *Example 4: Transfer Messages from Server to Client* under *Examples for EntireX Broker Tutorial*
  - *Example 5: Transfer Messages from Client to Server* under *Examples for EntireX Broker Tutorial*

Note: See *Examples for EntireX Broker Tutorial*.

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* in the administration documentation.

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
```

Example: Programming Language Natural

- *Example 7: Send Messages with HOLD - Delayed Delivery* under *Examples for EntireX Broker Tutorial*

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* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

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

- PUBLISHER-NONACT
- SUBSCRIBER-NONACT
- 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. This fact must be considered where an application component implements both publish and subscribe and client and server.

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* in the general administration documentation.

Unit of Work Lifetime

The UWTIME parameter in the *Broker Attributes* in the administration documentation 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* in the administration documentation:

- 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* in the ACI Programming documentation. 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* in the ACI Programming documentation.

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* in the general administration documentation. 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 Progamming 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 Internationalization

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

This section covers the following topics:

- General Information
- Providing Locale Strings
- Using the ENVIRONMENT Field with the Translation User Exit

General Information

The broker stub does not convert your application data before it is sent to the broker. The application's data is shipped as given.

For the internationalization approaches ICU conversion and SAGTRPC user exit, valid locale strings are required for conversion to behave correctly.

Providing Locale Strings

Under the Windows operating system:

- The broker stub assumes by default that 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 and communicating with a broker version 7.2.*n* or above, a codepage identifier of this Windows ANSI codepage is also automatically transferred as part of the locale string to tell the broker how the data is encoded.
- If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation.

Under all other operating systems:

- The broker stub does not automatically send a codepage identifier to the broker as part of the locale string.
- The broker stub assumes the broker's locale string defaults match. If they do not match, provide the codepage explicitly. See *Broker's Locale String Defaults* under *Locale String Mapping* in the internationalization documentation.

With the ACI control block field `LOCALE-STRING`:

- You can override or provide a codepage in the locale string sent to the broker. If a codepage is provided, it must follow the rules described under *Locale String Mapping* in the internationalization documentation.

- You can force a locale string to be sent if communicating with broker version 7.1.x and below. Under Windows you can use the abstract codepage name. See *Using the Abstract Codepage Name LOCAL* under *Locale String Mapping* in the internationalization documentation.
- API version 4 or above is required to override the locale string.

The encoding in which your application gives the data to the broker stub and the locale string

- must always match, i.e. the codepage derived after the broker's built-in locale string mapping process must be the same as the encoding of the data provided. See *Broker's Built-in Locale String Mapping* under *Locale String Mapping* in the internationalization documentation.
- must be a codepage supported by the broker, depending on the internationalization approach;

otherwise, unpredictable results will occur.

Example for Assembler

```
MVC   S$LOCALE,=C'ECS037'                 MOVE CP
....
```

Examples for C

1. Using a specific codepage

   ```
   /* prepare the locale-string with a codepage */
   memset (pETBCB->locale_string,' ',sizeof(pETBCB->locale_string));
   strncpy(pETBCB->locale_string,"ECS0819",sizeof(pETBCB->locale_string));
   ....
   ```

2. Using the platform's default codepage (Windows only)

   ```
   /* prepare the locale-string with a codepage */
   memset (pETBCB->locale_string,' ',sizeof(pETBCB->locale_string));
   strncpy(pETBCB->locale_string,
   ETB_CODEPAGE_USE_PLATFORM_DEFAULT,sizeof(pETBCB->locale_string));
   ....
   ```

Example for COBOL

```
MOVE 'ECS037' TO LOCALE-STRING.
....
```

Examples for Natural

```
MOVE 'ECS037' TO #SDPA-API.#LOCALE_STRING.
....
```

Using the ENVIRONMENT Field with the Translation User Exit

Using the internationalization approach *translation user exit*, an ACI programmer can provide additional information to their translation exit through the `ENVIRONMENT` field, allowing flexible translation behavior in accordance with application requirements. The field cannot be used for any other internationalization approaches and must be empty if a method other than translation user exit is used. See *Translation User Exit* under *Introduction to Internationalization*.

Example

Assume a broker service or topic has a user-written translation routine called ABCTRAN, which is capable of performing several types of data conversion, for example EBCDIC-ASCII translation, byte swapping, and mixed data types. The user translation routine may need to know the data formats used by both partners. The `ENVIRONMENT` field can be used to pass this information from the application to the translation routine in Broker kernel.

Technique

```
MOVE 'MYCODEPAGE' TO #ETBCB.#ENVIRONMENT
...
CALL 'BROKER' #ETBAPI #SEND-BUFF #RECV-BUFF #ERR-TXT
```

Using Send and Receive Buffers

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` and `RECEIVE_PUBLICATION` functions 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

Conversion and translation 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 internationalization approach. See *Internationalization with EntireX*. Note the following:

For translation and 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 or RECEIVE_PUBLICATION 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 or RECEIVE_PUBLICATION 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. **Note:** Under z/OS with Adabas version 8, the value for NET is the same as for TCP and SSL.
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 *Administration of Broker Stubs under z/OS* in the z/OS administration documentation.

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 *Administration of Broker Stubs under z/OS* in the z/OS administration documentation.

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

Transport Methods

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) (6)
z/OS (2)	Batch, TSO, IMS (BMP)	BROKER	x	x	x	
	Com-plete	COMETB	x	(3)	x	
	CICS	CICSETB	x	(3)	x	
	IMS (MPP)	MPPETB	x	x	x	
	IDMS/DC (4)	IDMS	x	(3)		
	Natural	NATETB23	x	x	x	
	UNIX System Services	*Java ACI* in the Developer's Kit documentation	x	x		x
UNIX		broker.so	x	x		
		Java ACI in the Developer's Kit documentation	x	x		x
Windows		broker.dll (5)	x	x		
		Java ACI in the Developer's Kit documentation	x	x		x
BS2000/OSD	Batch, Dialog (formerly TIAM)	BROKER	x	x	x	
z/VM		BKIMBCMS	x		x	
IBM i		EXA	x			
OpenVMS		BROKER	x	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 you can use IBM's Application Transparent Transport Layer Security (AT-TLS) as an alternative to direct SSL support inside the broker stub. Refer to the IBM documentation for more information.
3. Use AT-TLS. See Note 2.
4. Tracing and transport timeout are not supported in this environment.

5. Stub broker32.dll is supported for reasons of backward compatibility. The functionality is identical to broker.dll.
6. Via Broker HTTP(S) Agent; see *Settting up and Administering the Broker HTTP(S) Agent* in the UNIX and Windows administration documentation.

See also:

- *Setting 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 *Setting 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 and TLS

Application programs using Secure Sockets Layer (SSL) or Transport Layer Security (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 or TLS transport. See `SETSSLPARMS` and *Running Broker with SSL or TLS Transport* in the platform-specific administration documentation.

Example: An application communicating through SSL or TLS would specify on each command the Broker ID:

```
MYPC:1958:SSL
```

where the host on which the Broker kernel executes is known to SSL or TLS as MYPC and is listening on port 1958.

Transport Examples

- **For programming language C under Windows:**

  ```
  strcpy( pSBuf, "TRUST_STORE=c:\\certs\\CaCert.pem&VERIFY_SERVER=N");
  EtbCb.send_length = strlen(pSBuf);
  EtbCb.errtext_length = 40;
  EtbCb.function = FCT_SETSSLPARMS
  rc = broker (etbcb, pSBuf, (char *) 0, pEBuf);
  ```

- **For programming language Natural under z/OS:**

  ```
  MOVE 'TRUST_STORE=UID/KEYRING' TO #SSL-BUFF
  MOVE 80 TO #ETBCB.#SEND-LENGTH MOVE 40 TO #ETBCB.#ERRTEXT-LENGTH
  MOVE #FCT-SSLP TO #ETBCB.#FUNCTION
  MOVE 'IBMHOST:1958:SSL' TO #ETBCB.#BROKER-ID
  ...
  CALL 'BROKER' #ETBAPI #SSL-BUFF #RECV-BUFF #ERR-TXT
  ```

See table above for how SSL or TLS is supported depending on broker stub and platform.

For information on Secure Sockets Layer, see *SSL or TLS and Certificates with EntireX*.

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` or `RECEIVE_PUBLICATION` 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* in the ACI Programming documentation.

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.

IAF Authentication

With ACI field `CREDENTIALS-TYPE` you can specify that the token specified in the IAF token field is used to authenticate the user. If this field is left blank, user ID and password will be used as before.

4 Writing Applications: Units of Work

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 Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

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

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

<table>
<tr><th>Function</th><th>Fields in EntireX Broker Control Block</th></tr>
<tr><td>DEREGISTER</td><td>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]</td></tr>
<tr><td>RECEIVE</td><td>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]</td></tr>
<tr><td>REGISTER</td><td>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</td></tr>
<tr><td>SEND</td><td>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</td></tr>
</table>

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* under *Writing Applications: Units of Work* in the ACI Programming documentation.
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* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.

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* under *Concepts of Persistent Messaging* in the general administration documentation.

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)

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

Client	Server

`>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.	
`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* under *Concepts of Persistent Messaging* in the general administration documentation.

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.	

`>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.

Client	Server

	`>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.	
`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

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. See example under *Attach Manager Interface* under *Examples for EntireX Broker Tutorial*. 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).

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

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* in the ACI Programming documentation.
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* in the ACI Programming documentation.
SERVICE=CMD	Specify this for the command service.
SERVICE=PARTICIPANT-SHUTDOWN	Specify this for the participant shut-down 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 under *Broker CIS Data Structures* in the ACI Programming documentation	Input		Used by an application to specify an information service request.
Command Request Structure under *Broker CIS Data Structures* in the ACI Programming documentation		Input	Used by an application to specify a command service request.
Common Header Structure for Response Data under *Broker CIS Data Structures* in the ACI Programming documentation	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 ■ WORKER ■ CLIENT ■ SERVER ■ CONVERSATION ■ PSF ■ PSFDIV ■ PSFADA ■ PSFFILE ■ SUBSCRIBER ■ PUBLISHER ■ PUBLICATION ■ TOPIC ■ CMDLOG-FILTER ■ NET ■ PSFCTREE ■ SECURITY ■ SSL ■ TCP

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* under *Broker CIS Data Structures* in the ACI Programming documentation.

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* under *Broker CIS Data Structures* in the ACI Programming documentation. For each object for which information is returned, there is one information reply structure containing the information.

Send Buffer	*Information Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation
Receive Buffer	*Common Header Structure for Response Data* under *Broker CIS Data Structures* in the ACI Programming documentation [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* under *Error Messages and 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* under *Broker CIS Data Structures* in the ACI Programming documentation. When sending a command service request, note the possible combinations under *Command Request Parameter Combinations* under *Broker CIS Data Structures* in the ACI Programming documentation.

The return data in the receive buffer includes the common header structure (see *Common Header Structure for Response Data* under *Broker CIS Data Structures* in the ACI Programming documentation):

Send Buffer	*Command Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation
Receive Buffer	*Common Header Structure for Response Data* under *Broker CIS Data Structures* in the ACI Programming documentation

Tips

- The error code in the common header structure must be tested by the application programmer. See *Broker Command and Information Services Error Codes* under *Error Messages and 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

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* in the EntireX Security documentation and *Writing Security Exits* under *Using Sample Security Exits for Broker Security*.

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` under *Broker CIS Data Structures* in the ACI Programming documentation.

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` under *Broker CIS Data Structures* in the ACI Programming documentation. 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` under *Broker CIS Data Structures* in the ACI Programming documentation.

7 Writing Applications using EntireX Security

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*.

Note: ACI versions 1-7 apply to the communication model client and server only. ACI version 8 and above apply to the communication models client and server and publish and subscribe.

General Programming Considerations

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

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

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`, `SERVER-NONACT`, `PUBLISHER-NONACT` or `SUBSCRIBER-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* under *Configuration Options for Broker* in the EntireX Security documentation.

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* under *Configuration Options for Broker* in the EntireX Security documentation.

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* under *Configuration Options for Broker* in the EntireX Security documentation.

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 *Configuration Options for Broker* in the EntireX Security documentation.

FORCE-LOGON

This topic does not apply to the publish-and-subscribe communication model since this requires an explicit logon and cannot use `AUTOLOGON`.

`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* under *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

`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.

IAF Tokens

The Integrated Authentication Framework (IAF) is a token-based infrastructure that enables Software AG's enterprise single sign-on. In addition, it allows usage of a configurable authentication system (user database) with Software AG products across platforms. IAF is part of the Software AG Security Infrastructure.

Authorization

Publish and Subscribe

Applications that create publications are subject to authorization requests under EntireX Security. For every new publication, an authorization check is performed based on the `TOPIC`. Publications are transmitted to subscriber applications only if the authorization check is successful; otherwise an ACI response is returned to the application issuing the `SEND_PUBLICATION` command.

Subscriber applications are subject to an authorization check if security is installed for EntireX Broker. An authorization check based on the topic is performed when the subscriber application issues a `SUBSCRIBE` command. The application is allowed to subscribe only if the authorization check is successful; otherwise an ACI response code is returned to the subscriber. Similarly, if the administrator performs third-party subscription or unsubscription on behalf of a subscriber using command and information services (CIS), an authorization check is made, based on the topic. See *Writing Applications: Command and Information Services* in the ACI documentation.

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

See also publish-and-subscribe example under *Resource Profiles in EntireX Security* in the EntireX Security documentation and *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

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* under *Writing Applications using EntireX Security* in the ACI Programming documentation.

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* under *Writing Applications using EntireX Security* in the ACI Programming documentation. 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* in the EntireX Security documentation and *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.

Authorization Rules

For UNIX and Windows, Broker authorization checks are made using a set of definitions maintained in an LDAP repository (Lightweight Directory Access Protocol). Authorization rules are the mechanism by which authorization checks are performed for UNIX and Windows. For more information see *Configuring Authorization Rules* in the UNIX and Windows administration sections.

Encryption

Introduction

Encryption of the message data with the EntireX Security encryption is configured in the broker and in client and server applications. The `ENCRYPTION-LEVEL` attribute configured in the broker attribute file is a service-specific attribute. Allowed values are 0, 1, and 2. Using `ENCRYPTION-LEVEL` 1 or 2 requires `SECURITY=YES` in the `DEFAULTS = BROKER` section.

Encryption Level	Description
0	No encryption requested, but allowed.
1	Encryption for server requested, client can encrypt.
2	Encryption for client and server requested.

Example:

```
DEFAULTS = SERVICE
  CONV-NONACT      = 5M
  DEFERRED         = YES
  SERVER-NONACT    = 10M
  ENCRYPTION-LEVEL = 2
  CLASS = RPC,   SERVER = SRV1, SERVICE = CALLNAT, TRANSLATION = SAGTCHA
```

Applications can configure the encryption level by setting the broker ACI field ENCRYPTION-LEVEL or using the methods of the Java ACI. Allowed values are 0, 1, and 2. 0 does not encrypt the message, 1 encrypts the message to the broker, while 2 encrypts the message to the broker and requests that the partner application also encrypts the message. To guarantee end-to-end encryption from client to server, use encryption level 2.

The broker controls the encryption and issues one of the error codes 00200401 | 00200419 | 00200420 | 00200421 | 00200422 if encryption levels do not match.

In all cases encryption requires broker connections secured with EntireX Security. Broker returns error code 00210402 if this is violated.

Encryption behavior is slightly different depending on whether the application is Java-based or not. For non-Java-based applications, the behavior depends on the ACI version. These differences are described below.

Encryption for non-Java Applications

For non-Java applications, Encryption behavior depends on the ACI version:

- ACI Version 8 and Above
- ACI Version 6 and 7
- ACI Version 1 to 5

Note: In the sections below, "EL" refers to the field ENCRYPTION-LEVEL in the ACI control block.

ACI Version 8 and Above

Encryption is configured by the application per message. The table shows allowed combinations of ENCRYPTION-LEVEL and EL.

ENCRYPTION-LEVEL in Broker Attribute File	Server	Client
0 (or undefined)	EL= 0 or 1	EL= 0 or 1
1	EL= 1	EL= 0 or 1
2	EL= 2	EL= 2

ACI Version 6 and 7

Encryption is configured by installing the security exit for the application (security exit is `secuexit.dll` on windows, `secuexit.so` on UNIX, `SECUEXI0` on z/OS) and the environment variable `NA2_BKPRIV` (or `ETB_ENCRPYT`) is set to 1. The table shows allowed combinations of `ENCRYPTION-LEVEL` and `EL`.

`ENCRYPTION-LEVEL` in Broker Attribute File	Server	Client
0 (or undefined)	EL= 0	EL= 0 or 1
1	EL= 1	EL= 0 or 1
2	EL= 2	EL= 2

ACI Version 1 to 5

Encryption is configured by installing the security exit for the application (security exit is `secuexit.dll` on windows, `secuexit.so` on UNIX, `SECUEXI0` on z/OS) and the environment variable `NA2_BKPRIV` (or `ETB_ENCRPYT`) is set to 1. Only `ENCRYPTION-LEVEL` in the broker attribute file is available. Field `EL` for the stub is not available.

Encryption for Java-based Applications (ACI and RPC)

On the Broker object, use the `methods setSecurity(...)` or `useEntireXSecurity(...)` to enable EntireX Security and set the encryption level. See `Broker` in the Javadoc documentation of the Java ACI in the Javadoc for details.

RPC Servers

For RPC servers, the encryption level is set in the configuration. See documentation of the configuration for the specific RPC server.

RPC Clients

- **C-based RPC Clients**
  Use the structure `ERX_CLIENT_IDENTIFICATION` to set the encryption level. See *`ERX_CLIENT_IDENTIFICATION`* for details.
- **Natural RPC Clients**
  Natural Clients use user exit USR4009N to set the encryption level. See *Operating a Natural RPC Environment* in the Natural documentation.

8 Broker ACI Fields

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 diagnos data.
ADCOUNT	I4		2	O	A count of the number of times an attempt was made to deliver a UC 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* ur *Writing Applications: Client and Server | Publish and Subscribe* in the Programming documentation.
API-VERSION	I1	1-10	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-meth Style***. In order to communicate, applications must specify the sam BROKER-ID. **Note:** URL style does not apply to mainframe platforms (z/OS, BS2000/OSD and z/VSE).
CLIENT-ID	I4	1-2147483647	9	O	Returns to a server application the unique instance number of a cl 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 th returned to the server in the CLIENT-UID field. If EntireX installed, it is valid for the server application to rely on thi when making decisions concerning access to information. See *Authentication* under *Configuration Options for Broker* in Security documentation (z/OS only). **Note:** There is an uppercase translation when the USER-I propagated to the CLIENT-UID field under EntireX Securi 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* under *Writing Ap Client and Server | Publish and Subscribe* in the ACI Prograr documentation. The following values are possible: **0 - 9** — 0 = no compression, 9 = maxi 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 Bro and server must include the CONV-ID in their communicat and server can also specify the indicated textual values (ca order to indicate to Broker the expected status of the conv Messages for the conversation are taken from the queue o first-out basis. See *Conversational and Non-conversational M Writing Applications: Client and Server* in the EntireX Broke Programming documentation. **NEW** — On a SEND function, initiates a new co On a RECEIVE function, signals read receive requests for new conversatio CONV-ID value is assigned to the con and the value is returned to the calle **OLD** — Applies to RECEIVE function only. C messages for existing conversations a **ANY** — On a RECEIVE function, requests or m returned on a first-in, first-out basis conversation. On an EOC function, ar

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td><table>
<tr><td></td><td>conversations belonging to the caller are terminated.</td></tr>
<tr><td><b>NONE</b></td><td>On a <code>SEND</code> function, the message is non-conversational.</td></tr>
<tr><td><b>conv-id</b></td><td>Indicates a specific conversation.
The <code>CONV-ID</code> value is an internally generate identifier (containing numeric characters only alphanumeric characters) for the conversatio Application programmers are advised to ma no assumptions about the contents, layout, o meaning of any part of the <code>CONV-ID</code> field.
If the client has specified <code>API-VERSION</code> 3 or above, the <code>CONV-ID</code> contains both alphanume and numeric characters.
If the Broker does not support UOW process (the Broker attribute <code>MAX-UOWS=0</code>) or the clie has specified <code>API-VERSION</code> or <code>2</code>, the <code>CONV-I</code> contains numeric characters.</td></tr>
</table></td></tr>
<tr><td><code>CONV-STAT</code></td><td>I1</td><td>1 | 2 | 3</td><td>2</td><td>O</td><td>Conversation Status. See <i>Managing Conversation Contexts</i> under <i>Writ Applications: Client and Server</i> in the EntireX Broker ACI Programm documentation.<table>
<tr><td><b>1</b></td><td><code>NEW</code> - The message is the first in a new conversation.</td></tr>
<tr><td><b>2</b></td><td><code>OLD</code> - The message is part of an existing conversation.</td></tr>
<tr><td><b>3</b></td><td><code>NONE</code> - The message is non-conversation</td></tr>
</table></td></tr>
<tr><td><code>CREDENTIALS-TYPE</code></td><td>binary</td><td></td><td>9</td><td>O</td><td>Determines the credentials type to be used to authenticate a user.<table>
<tr><td><i><code>blank</code></i></td><td>Default. Use user ID and password.</td></tr>
<tr><td><b>IAF</b></td><td>The token specified in the IAF token fie is used.</td></tr>
</table></td></tr>
<tr><td><code>DATA-ARCH</code></td><td>I1</td><td></td><td>4</td><td>I</td><td>Architecture code. For future use.</td></tr>
<tr><td><code>ENCRYPTION-LEVEL</code></td><td>I1</td><td>0|1|2</td><td>6</td><td>I</td><td>Encryption level. See <i>Encryption</i> under <i>Writing Applications using EntireX Security</i>.</td></tr>
<tr><td><code>ENVIRONMENT</code></td><td>A32</td><td>string</td><td>1</td><td>I</td><td>Information for translation user exits.
The contents of the field are solely the responsibility of the applicat and its associated translation user exit. The field cannot be used for other internationalization approaches and must be empty if a met other than translation user exit is used. See <i>Using the <code>ENVIRONMEN</code></i></td></tr>
</table>

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td><i>Field with the Translation User Exit</i> under <i>Writing Application</i>
<i>Server | Publish and Subscribe</i> in the ACI Programming doc</td></tr>
<tr><td><code>ERROR-CODE</code></td><td>A8</td><td></td><td>1</td><td>O</td><td>Returns an error code to the caller. The application should
contents of this field at the completion of every Broker fu
<i>Error Handling</i> under <i>Writing Applications: Client and Serve</i>
<i>and Subscribe</i> in the ACI Programming documentation. Th
digits represent the error class; the next four digits repres
number; see also <i>Error Messages and Codes</i>.</td></tr>
<tr><td><code>ERRTEXT-LENGTH</code></td><td>I4</td><td>0-40 | 0-255</td><td>1 | 9</td><td>I</td><td>Length of the error text buffer in bytes. See <i>Call Format</i> in t
language-specific EntireX ACI documentation.

If there are fewer than 40 bytes, the error text may be trunca
of 0 (zero) means no error text.

<b>Note:</b> In previous ACI versions, Broker kernel always retu
of error text that were space-padded if necessary. With AC
and above, variable-length error texts can be returned to i
logging and tracing.</td></tr>
<tr><td><code>FORCE-LOGON</code></td><td>A1</td><td>Y | N</td><td>6</td><td>I</td><td>Override the <code>AUTOLOGON</code> feature of the Broker. See <code>AUTOL</code>
<table><tr><td><b>Y</b></td><td>The attribute <code>AUTOLOGON=YES</code> in the Bro
file is overridden. See <i><code>FORCE-LOGON</code></i> un
<i>Applications using EntireX Security</i> in the
Programming documentation.</td></tr><tr><td><b>N</b></td><td>Default. Use the value of the Broker att
for <code>AUTOLOGON</code>.</td></tr></table></td></tr>
<tr><td><code>FUNCTION</code></td><td>I1</td><td>1-22</td><td>1</td><td>I</td><td>The Broker function to be performed. A function value is r
is modified by the ACI field <code>OPTION</code> and the other ACI fi
below for description of values.
<table>
<tr><td><b>1</b></td><td><code>SEND</code></td><td><b>14</b></td><td><code>KERNELVERS</code></td></tr>
<tr><td><b>2</b></td><td><code>RECEIVE</code></td><td><b>15</b></td><td><code>LOCTRANS</code> (d</td></tr>
<tr><td><b>4</b></td><td><code>UNDO</code></td><td><b>16</b></td><td><code>SETSSLPARM</code></td></tr>
<tr><td><b>5</b></td><td><code>EOC</code></td><td><b>17</b></td><td><code>SEND_PUBLI</code></td></tr>
<tr><td><b>6</b></td><td><code>REGISTER</code></td><td><b>18</b></td><td><code>RECEIVE_PU</code></td></tr>
<tr><td><b>7</b></td><td><code>DEREGISTER</code></td><td><b>19</b></td><td><code>SUBSCRIBE</code></td></tr>
<tr><td><b>8</b></td><td><code>VERSION</code></td><td><b>20</b></td><td><code>UNSUBSCRIB</code></td></tr>
<tr><td><b>9</b></td><td><code>LOGON</code></td><td><b>21</b></td><td><code>CONTROL_PU</code></td></tr>
<tr><td><b>10</b></td><td><code>LOGOFF</code></td><td><b>22</b></td><td><code>REPLY_ERRO</code></td></tr>
<tr><td><b>13</b></td><td><code>SYNCPOINT</code></td><td></td><td></td></tr>
</table></td></tr>
<tr><td><code>KERNELSECURITY</code></td><td>A1</td><td>Y | U | N</td><td>7</td><td>I/O</td><td>This field is used by the application to indicate programm
intention to communicate with a secure/non-secure Brok</td></tr>
</table>

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>also indicates to the application how security has been configured a particular Broker kernel. See Broker attribute SECURITY.
When used as an <b>input</b> field, this field is used by programmer to indicate the desired security behavior of the application. If no opt is specified, defaults to administrator's configuration setup.<table><tr><td><b>Y</b></td><td>EntireX Security</td></tr><tr><td><b>U</b></td><td>User-written Security</td></tr><tr><td><b>N</b></td><td>No security</td></tr></table><b>Notes</b><ul><li><b>Output</b>
In version 7 or above, this field returns the output value when executing the KERNELVERSION command.</li><li><b>Input</b>
In version 8 or above, the application can programmatically spec the desired security behavior for all commands other than KERNELVERSION.</li></ul></td></tr>
<tr><td>LOCALE-STRING</td><td>A40</td><td>string</td><td>4</td><td>I</td><td>It is assumed that you have read the document <i>Internationalization w EntireX</i> and are familiar with the various internationalization approaches described there.
The locale string tells the broker the encoding of the data. No convers is done within the broker's stub. The application must ensure the c provided matches the locale string. The locale string is case-insensit also dashes '-' and underscores '_' are ignored (dashes and undersc improve human readability). See <i>Using Internationalization</i> under <i>Wri Applications: Client and Server | Publish and Subscribe</i> in the ACI Programming documentation.</td></tr>
<tr><td>LOG-COMMAND</td><td>I1</td><td>0 | 1</td><td>9</td><td>I</td><td>Components that communicate with Broker can trigger command logging by setting this field. By default, command logging is based the command log filters set in the kernel. You may override these ker settings programmatically by setting this LOG-COMMAND field. If th field is set, all associated commands will be logged.
<b>Note:</b> If command logging is not enabled for your kernel, you mus first contact your administrator.</td></tr>
<tr><td>LONG-BROKER-ID-LENGTH</td><td>I4</td><td>0-2147483647</td><td>10</td><td></td><td>Length of LONG-BROKER-ID. If the value is non-zero, specify the va 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:</td></tr>
</table>

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td><code>tcpip://[2001:0db8:85a3:08d3:1319:8a2e:0370:7</code>
<code>[2001:0db8:85a3:08d3:1319:8a2e:0370:7347]:393</code>
<code>(2001:0db8:85a3:08d3:1319:8a2e:0370:7347):393</code>
The IP address is enclosed in square brackets or parenthes</td></tr>
<tr><td>MSG-ID</td><td>B32</td><td></td><td>2</td><td>I/O</td><td>Not used by EntireX Broker.</td></tr>
<tr><td>MSG-TYPE</td><td>A16</td><td></td><td>2</td><td>I/O</td><td>Not used by EntireX Broker.</td></tr>
<tr><td>NEWPASSWORD</td><td>B32</td><td>Can contain binary data.</td><td>2</td><td>I</td><td>Specifies a new password to be transmitted to the Broker ker the authentication of the application. See <i>Authentication</i> ur <i>Configuration Options for Broker</i> in the EntireX Security docu
The current password can be changed only when the clier authenticates itself. This occurs on the first Broker ACI fur be LOGON) and requires the application to assign to the Bro fields PASSWORD and NEWPASSWORD.</td></tr>
<tr><td>OPTION</td><td>I1</td><td>0-21</td><td>1</td><td>I</td><td>Provides additional information that modifies the behavio Broker ACI functions.
<table>
<tr><td><b>0</b></td><td>no option</td><td><b>8</b></td><td>NEXT</td><td><b>16</b></td><td>QUE</td></tr>
<tr><td><b>1</b></td><td>MSG</td><td><b>9</b></td><td>PREVIEW</td><td><b>17</b></td><td>SET</td></tr>
<tr><td><b>2</b></td><td>HOLD</td><td><b>10</b></td><td>COMMIT</td><td><b>18</b></td><td>ANY</td></tr>
<tr><td><b>3</b></td><td>IMMED</td><td><b>11</b></td><td>BACKOUT</td><td><b>19</b></td><td>rese futu</td></tr>
<tr><td><b>4</b></td><td>QUIESCE</td><td><b>12</b></td><td>SYNC</td><td><b>20</b></td><td>DUR</td></tr>
<tr><td><b>5</b></td><td>EOC</td><td><b>13</b></td><td>ATTACH</td><td><b>21</b></td><td>CHE</td></tr>
<tr><td><b>6</b></td><td>CANCEL</td><td><b>14</b></td><td>DELETE</td><td></td><td></td></tr>
<tr><td><b>7</b></td><td>LAST</td><td><b>15</b></td><td>EOCCANCEL</td><td></td><td></td></tr>
</table></td></tr>
<tr><td>PARTNER-BROKER-ID</td><td>A32</td><td>string</td><td>9</td><td>O</td><td>ID of the partner broker. Deprecated.</td></tr>
<tr><td>PASSWORD</td><td>A32</td><td>Can contain binary data.</td><td>1</td><td>I</td><td>Specifies a password to be transmitted to the Broker to che authentication of the application. See <i>Authentication</i> under C <i>Options for Broker</i> in the EntireX Security documentation a <i>FORCE-LOGON</i> under <i>Writing Applications using EntireX Sec</i> ACI Programming documentation.</td></tr>
<tr><td>PTIME</td><td>A8</td><td></td><td>2</td><td>I</td><td>Not used by EntireX Broker.</td></tr>
<tr><td>PUBLICATION-ID</td><td>A16</td><td>string, case-sensitive.</td><td>8</td><td>I/O</td><td>Publication ID for publish-and-subscribe communication
A unique ID assigned to each publication by EntireX Broke and subscribers must include the publication ID and the (their communications. Publisher and subscriber can also s indicated textual value (capitals) in order to indicate to Br expected status of the publication. Messages for the publi queued to the topic on a first-in, first-out basis.</td></tr>
</table>

ACI Field	Format	Possible Values	API Vers	I/O	Description
					NEW: On a `SEND_PUBLICATION` function, initia a new publication. On a `RECEIVE_PUBLICATION` function, signal readiness to obtain next available publicati A publication ID value is assigned to the publication, and the value is returned to caller. **publication-id**: Indicates a specific publication. The `PUBLICATION-ID` value is an internally generated identifier (containing alphanumeric characters) for the publicati Application programmers are advised to make no assumptions about the content, layout or meaning of any part of the `PUBLICATION-ID` field.
`RECEIVE-LENGTH`	B32	Binary.	1	I/O	Specifies the length of receive buffer, in bytes. The maximum leng 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 sa as for TCP and SSL. See *Using Send and Receive Buffers* under *Writing Applications: Client Server | Publish and Subscribe* in the ACI Programming documentat
`RETURN-LENGTH`	I4		1	O	Length, in bytes, of the data returned. See *Using Send and Receive Buffers* under *Writing Applications: Client Server | Publish and Subscribe* in the ACI Programming documentat
`SECURITY-TOKEN`	B32	binary	1	I/O	The contents of this field depend heavily on the implementation of security exits. This field is utilized by EntireX Security. The application must main `SECURITY-TOKEN` between commands and not change this value. recommend that the application allocate a separate ACI control bl for each user if it issues commands on behalf of more than one use For objects executing inside Web servers, assigning a unique value such as 'session ID', to the ACI `TOKEN` field is highly recommende ensure uniqueness of user at same physical location. See *Ignore Secu Token* under *Configuration Options for Broker* in the EntireX Security documentation.

ACI Field	Format	Possible Values	API Vers	I/O	Description
					If EntireX Security is not implemented, and you choose to own security exits your can transmit an initial value to you exit as a credential that is used to calculate the actual secu After an application's authenticity has been verified by the exits, the `SECURITY-TOKEN` can be used to avoid additiona authentication checks.
`SEND-LENGTH`	B32	binary	1	I/O	Specifies the length of data being sent, in bytes. The maxin 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 as for TCP and SSL. See *Using Send and Receive Buffers* under *Writing Application Server | Publish and Subscribe* in the ACI Programming docu

`SERVER-CLASS`
`SERVER-NAME`
`SERVICE`
	A32 each	string, case-sensitive	1	I/O	A client uses these fields to identify the service that it requir uses this field to offer a service.
					Using all three fields allows you to organize servers, maki easier to identify, monitor, and maintain. Servers can be org server-classes, with each server providing a number of differ Each service must be defined in the attribute file (see *Servi Attributes* (`DEFAULTS=SERVICE`) under *Broker Attributes* in administration documentation).
					The service fields are required with `SEND`, `RECEIVE`, and `E` when `CONV-ID` is set to `NEW`, `OLD`, or `ANY`. When a `CONV-ID` the service fields are ignored.
					`SERVICE=*` or `SERVER-NAME=*` can be used on a `RECEIVE` indicate all services within a specified server or all servers specified server class.
					Note: Server classes "SAG", "Entire", "Adabas", "Natural", " and Broker are reserved for Software AG. Do not use them 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 fi function for a UOW to specify whether the UOW is persis Once established, the persistence of a UOW cannot be alte **0** none - Defaults to the value s the service.

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td><table><tr><td><b>1</b></td><td>OFF - The UOW is not persistent.</td></tr><tr><td><b>2</b></td><td>BROKER - The UOW is persistent.</td></tr></table></td></tr>
<tr><td>TOKEN</td><td>A32</td><td>string, case-sensitive.</td><td>1</td><td>I</td><td>Optionally identifies the caller and, when used, is required for all Broker ACI functions except VERSION. See <i>USER-ID and TOKEN</i> un <i>Writing Applications: Client and Server | Publish and Subscribe</i> in the <i>A</i> Programming documentation.
<b>Caution:</b> USER-ID and TOKEN must be specified by all application that use UOWs held in the persistent store, and by all publisher ar subscriber applications where publication and subscription data i held in the persistent store.</td></tr>
<tr><td>TOPIC</td><td>A96</td><td>string, case-sensitive</td><td>8</td><td>I/O</td><td>Topic name for publish and subscribe communication model.
A publisher uses this field to identify the topic name required. A subscriber uses this field to indicate the topic from which publicati are to be obtained. Each topic must be defined in the attribute file. <i>Topic-specific Attributes</i> (DEFAULTS=TOPIC) under <i>Broker Attributes</i> the administration documentation.</td></tr>
<tr><td>UOWID</td><td>A16</td><td></td><td>3</td><td>I/O</td><td>A unique identifier for a UOW.
The value is returned on the first SEND or RECEIVE command wit a UOW; the value must be provided on all subsequent SEND, RECEI and SYNCPOINT commands related to the same UOW. Client and ser can also specify the indicated textual value (capitals) in order to indic to Broker the following:<table><tr><td><b>BOTH</b></td><td>Since a server receives a UOW and rep with a different UOW, both UOWs can committed or backed out by specifying UOWID=BOTH for the SYNCPOINT comma</td></tr><tr><td><i>uowid</i></td><td>The <i>uowid</i> must be supplied in subsequ SEND, RECEIVE and SYNCPOINT comman related to the same UOW.</td></tr></table></td></tr>
<tr><td>UOWSTATUS</td><td>I1</td><td></td><td>3</td><td>O</td><td>Contains the status of a UOW. EntireX Broker returns the UOWSTAT field to the calling application in order to provide information abo the condition of the specified UOW.<table><tr><td><b>1</b></td><td>RECEIVED - One or more messages have b sent as part of a UOW but the UOW has n yet been committed.</td></tr><tr><td><b>2</b></td><td>ACCEPTED - The UOW has been committe by the sender.</td></tr><tr><td><b>3</b></td><td>DELIVERED - The UOW is currently being received.</td></tr></table></td></tr>
</table>

ACI Field	Format	Possible Values	API Vers	I/O	Description
					4 BACKEDOUT - The UOW has been by the sender. **5** PROCESSED - The UOW has been r the receiver has committed it. **6** CANCELLED - The UOW has been c the receiver. **7** TIMEOUT - The UOW was not pro within the time allowed. **8** DISCARDED - The UOW was not p and its data was discarded as the restart. With the exception of DELIVERED, all UOWSTATUS values ar Persistent values are kept until they are explicitly deleted or the time limit is exceeded. The lifetime of the UOWSTAT determined by the broker attribute UWSTATP. UOWSTATUS values in the following table are returned on function to indicate whether the message being transferre a UOW and, if so, its sequence within the UOW: **0** NONE - The message is not part of **9** FIRST - The message is the first m UOW. **10** MIDDLE - The message is neither t the last in the UOW. **11** LAST - The message is the last me UOW. **12** ONLY - The message is the only me UOW.
UOW-STATUS-PERSIST		0 - 255	3	I	The value of the UOW-STATUS-PERSIST field is used as a to calculate the lifetime for the persistent status of a UOW is multiplied by the value of the broker attribute UWTIME. 255 can be specified to indicate no persistent status. **0** Means that the multiplier will same value as the UWSTATP Br attribute. **255** Means that there will be no pe status for UOWs. **1-254** Any number in this range is a multiplier.

ACI Field	Format	Possible Values	API Vers	I/O	Description
USER-DATA	B16	binary	2	I/O	Conversation User Data. See *Managing Conversation Contexts* unde *Writing Applications: Client and Server* in the EntireX Broker ACI Programming documentation.
USER-ID	A32	string, case-sensitive	1	I	Identifies the caller and is required for all Broker ACI functions exc VERSION. See *USER-ID and TOKEN* under *Writing Applications: Clie and Server* | *Publish and Subscribe* in the ACI Programming documentation. **Caution:** USER-ID and TOKEN must be specified by all publisher a subscriber applications where publication and subscription data i held in the persistent store.
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 retur to applications that query the status of the UOW. To update the USTA 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 va of this field determines how long the UOW status is to be retained the persistent store after the UOW is processed or timed out if it is processed. This is an alternative to specifying UOW-STATUS-PERSI UWSTAT-LIFETIME is specified on the first SEND_PUBLICATION function for a UOW; it is not allowed on a RECEIVE_PUBLICATIO function. ***n*S** The number of additional seconds th UOW status will exist. ***n*M** The number of additional minutes th UOW status will exist. ***n*H** The number of additional hours the UOW status will exist. ***n*D** The number of additional days the UO status will exist.
UWTIME	A8	*n*S | *n*M | *n*H | *n*D	3	I	The lifetime of a UOW. The UOW exists until its lifetime expires o is explicitly cancelled or backed out with SYNCPOINT OPTION=CANC 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 TIMEO UWTIME is specified on the first SEND function for a UOW; it is not allowed on a RECEIVE function.

<table>
<tr><th>ACI Field</th><th>Format</th><th>Possible Values</th><th>API Vers</th><th>I/O</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td><table>
<tr><td><b><i>n</i>S</b></td><td>The number of seconds the
exist.</td></tr>
<tr><td><b><i>n</i>M</b></td><td>The number of minutes the
exist.</td></tr>
<tr><td><b><i>n</i>H</b></td><td>The number of hours the U
exist.</td></tr>
<tr><td><b><i>n</i>D</b></td><td>The number of days the UO</td></tr>
</table></td></tr>
<tr><td>VARLIST-OFFSET</td><td>I4</td><td>0-2147483647</td><td>10</td><td>I</td><td>For Software AG internal use only.</td></tr>
<tr><td>WAIT</td><td>A8</td><td>NO | YES | <i>n</i>S | <i>n</i>M | <i>n</i>H</td><td>1</td><td>I</td><td>When a WAIT value (other than NO) is specified on a SEND
function, the caller will wait for a reply until the message
or the specified time limit has been reached. See <i>Blocked and</i>
<i>Broker Calls</i> under <i>Writing Applications: Client and Server |</i>
<i>Subscribe</i> in the ACI Programming documentation.<table>
<tr><td><b>NO</b></td><td>Default. No wait. Control is returne
caller.</td></tr>
<tr><td><b><i>n</i>S</b></td><td>The number of seconds the caller wi
reply.</td></tr>
<tr><td><b><i>n</i>M</b></td><td>The number of minutes the caller wi
reply.</td></tr>
<tr><td><b><i>n</i>H</b></td><td>The number of hours the caller will
reply.</td></tr>
<tr><td><b>YES</b></td><td>Depending on the role of the user (cl
publisher or subscriber), the respecti
is used (CLIENT-NONACT | SERVER-
PUBLISHER-NONACT | SUBSCRIBER
If a server registers multiple services,
value of all the services registered is
wait time for the server. However, if
both client and server, CLIENT-NON
used for calculating the wait time.

All different roles provide non-activit
The maximum value is taken for the</td></tr>
</table></td></tr>
</table>

9 Broker ACI Functions

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* in the ACI Programming documentation.

Overview Table

Function Name	Applicable Participant				Logon Required [1]	User ID Required [2]	Minimum API Version
	Client	Server	Publisher	Subscriber			
CONTROL_PUBLICATION			X	X	X	X	8
DEREGISTER		X				X	1
EOC	X	X				X	1
KERNELVERS	X	X	X	X		X	4
LOGOFF	X	X	X	X	X	X	2
LOGON	X	X	X	X		X	2
RECEIVE	X	X			[3]	X	1
RECEIVE_PUBLICATION				X	X	X	8
REGISTER		X				X	1
REPLY_ERROR	X	X				X	8
SEND	X	X			[3]	X	1
SEND_PUBLICATION			X		X	X	8
SETSSLPARMS	X	X	X	X			6
SUBSCRIBE				X	X	X	8
SYNCPOINT	X	X			X[3]	X	3
UNDO	X	X			X	X	2
UNSUBSCRIBE				X	X	X	8
VERSION	X	X	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

CONTROL_PUBLICATION

This function is used to control a publication.

Option	Description
BACKOUT	■ Used by the publisher, it backs out a publication in *received* status. ■ Used by the subscriber, it backs out the subscriber's reading of the publication; RECEIVE_PUBLICATION, PUBLICATION-ID=NEW will read it again.
COMMIT	■ Used by the publisher, it completes the publication, places it in *accepted* status and makes it available to a subscriber. ■ Used by the subscriber, it acknowledges receipt of a publication.
QUERY	Used by the publisher, it returns the status of the publication specified by the PUBLICATION-ID.
SETUSTATUS	Used by the publisher, it updates the optional application-specific user status of the specified publication that is in *received* status.

DEREGISTER

This function is used by a server application to deregister a service from EntireX Broker. Assigned resources are de-allocated. 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* under *Concepts of Persistent Messaging* in the general administration documentation.

- 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`.

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`* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation). 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`, `SEND_PUBLICATION`, `RECEIVE`, `RECEIVE_PUBLICATION`, `DEREGISTER` or `UNSUBSCRIBE` has been executed. It releases all resources used by the application immediately rather than waiting until they time out (see *Timeout Parameters* under *Writing Applications: Client and Server | Publish and Subscribe* in the ACI Programming documentation.

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 unless a `LOCTRANS` function (deprecated) is issued first to determine the `BROKER-ID`.

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.

Option	Description	Note
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* under *Concepts of Persistent Messaging* in the general administration documentation.	
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* under *Concepts of Persistent Messaging* in the general administration documentation.	

RECEIVE_PUBLICATION

This function is used by subscribers to receive publications. If PUBLICATION-ID=NEW is specified, Broker will deliver the next publication in *accepted* status to the subscriber, if a publication is in *accepted* status. In the case of subsequent RECEIVE_PUBLICATION commands, a valid PUBLICATION-ID is used to receive any messages from a multi-message publication. See function CONTROL_PUBLICATION to acknowledge the publication.

ACI Field	Description
RECEIVE-LENGTH	This field is required.
WAIT	You can specify a WAIT time to wait for a new publication.

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* in the administration documentation, 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 - e.g., 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.

SEND_PUBLICATION

This function is used by publishers to create publications for a specified topic. PUBLICATION-ID=NEW is used to create a new publication. In the case of a subsequent SEND_PUBLICATION, a valid PUBLICATION-ID is used when sending a multi-message publication. The publisher completes the publication either by specifying SEND_PUBLICATION,OPTION=COMMIT or using the function CONTROL_PUBLICATION. If SEND_PUBLICATION,OPTION=COMMIT is used, the option is specified on the final SEND_PUBLICATION in the sequence.

Option/ ACI Field	Description
OPTION=COMMIT	Required to complete a publication if the function CONTROL_PUBLICATION is not used.
PUBLICATION-ID=NEW	Required to start a new publication.
SEND-LENGTH	Required.

SETSSLPARMS

This function is used by clients and servers to set the SSL parameters. It can be called whenever the transport method SSL is available.

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 *Running Broker with SSL or TLS Transport* in the platform-specific administration documentation.

SUBSCRIBE

This function is used to request EntireX Broker to subscribe to a specific topic. The *Topic-specific Attributes* (DEFAULTS=TOPIC) under *Broker Attributes* in the administration documentation contains parameters that specify the behavior of the topic. If Broker runs with the setting SECURITY=YES in the Broker attribute file, the user must be successfully authenticated by EntireX Broker and also authorized to subscribe to the specific topic.

The TOPIC name must be defined in the Broker attribute file.

Option	Description
DURABLE	If durable is specified within the SUBSCRIBE command, users need to subscribe only once to a topic. The subscription is retained after the user issues a logoff command or if the subscriber has timed out. Similarly, the subscription remains if Broker is restarted. All publications necessary to meet subscription requirements are also retained. If a subscriber no longer wishes to subscribe to a topic, it must issue an UNSUBSCRIBE command; otherwise the subscription remains valid until the subscription expiration time has passed. Durable subscription requires the administrator to configure the persistent store. See *Concepts of Persistent Messaging* in the general administration documentation. In addition, the topic must be specified as durable within the *Topic-specific Attributes* (DEFAULTS=TOPIC) under *Broker Attributes* in the administration documentation.

ALLOW-DURABLE is a topic-specific attribute that determines whether a subscriber is allowed to perform a durable subscription to a topic. If users are allowed to durably subscribe to any topic, you must specify a value for the SUBSCRIBER-STORE attribute.

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* in the general administration documentation.

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* under *Concepts of Persistent Messaging* in the general administration documentation.
`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.

Option	Description
`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 UOW sent by the caller. In addition, `SERVER-CLASS`, `SERVER-NAME` and `SERVICE` details of the associated server are also returned. The `CONV-ID` can be included to qualify the request.
`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* in the general administration documentation.

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

UNSUBSCRIBE

This function is used to cancel the subscription to a specific topic. The fully specified topic name is required to execute the `UNSUBSCRIBE` request.

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	Description
1	MSG	Used with a RECEIVE function to receive only a message that is not part of a UOW.
2	HOLD	Used in conversational mode only. ■ On a SEND function, places the messages in a HOLD queue. Messages are released by a SEND without HOLD. ■ On an UNDO function, releases all previously held messages. ■ On a LOGOFF function, the conversation is not ended although the user is logged off.
3	IMMED	Used with the DEREGISTER function to immediately terminate all conversations for the specified server. All partners are informed with an appropriate error code.
4	QUIESCE	Used with the DEREGISTER function to terminate a server smoothly. Existing conversations are allowed to end normally; no new conversations are accepted. The server is removed from the "active" list. QUIESCE is the default option for the DEREGISTER function.
5	EOC	Used with the SEND function to 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* under *Concepts of Persistent Messaging* in the general administration documentation.
6	CANCEL	Used with the EOC function to abort a conversation rather than terminate normally. The receiver of a UOW can use SYNCPOINT OPTION=CANCEL to interrupt processing and discard the UOW.
7	LAST	■ Used in conversational mode with the RECEIVE function to retrieve the last (most recent) message. ■ Used with the SYNCPOINT function, it returns the status of the last UOW sent by the caller. ■ Used with the CONTROL_PUBLICATION function, it returns the status of the last publication sent by the publisher.
8	NEXT	Used with the RECEIVE function to retrieve the next unprocessed request or message. The request or message is then considered processed and can be accessed only with OPTION=LAST. NEXT is the default option for the RECEIVE function.
9	PREVIEW	Used with the RECEIVE function to 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	■ COMMIT is used with the SYNCPOINT function to commit the active UOW. For a UOW being sent, it means that the UOW is complete and can now be delivered

Number	Option	Description
		to the intended receiver. For a UOW being received, it means that that UOW is complete and no further processing of the UOW is allowed. ■ `COMMIT` is used with the `SEND` function to commit the active UOW. ■ `COMMIT` is used with the `CONTROL_PUBLICATION` function as follows: ■ Used by the publisher, it completes the publication and makes it available to subscribers. ■ Used by the subscriber, it acknowledges receipt of a read publication. ■ `COMMIT` is used with the `SEND_PUBLICATION` function to commit the publication and make it available to subscribers.
11	`BACKOUT`	■ The receiver of a UOW can use `SYNCPOINT` OPTION=BACKOUT to return the UOW to its undelivered state. The UOW can be processed again, in its entirety, by subsequent `RECEIVE` operations. The sender of a UOW can use `SYNCPOINT OPTION=BACKOUT` to delete the UOW. No further processing of the UOW is allowed. ■ `BACKOUT` is used with the `CONTROL_PUBLICATION` function as follows: ■ Used by the publisher, it back outs a publication. ■ Used by the subscriber, it back outs the subscriber's reading of the publication; `RECEIVE_PUBLICATION` will read it again.
12	`SYNC`	On a `SEND` function, indicates that the message is part of a UOW. On a `RECEIVE` function, indicates that the `RECEIVE` will be satisfied only by a message that is part of a UOW.
13	`ATTACH`	Used with the `REGISTER` function to register an attach server.
14	`DELETE`	Used with the `SYNCPOINT` function to delete the persistent status information for the specified UOW.
15	`EOCCANCEL`	Used with the `SYNCPOINT` function to cancel the conversation after committing a UOW.
16	`QUERY`	■ Used with the `SYNCPOINT` function only to query the status of a UOW. ■ Used with the `CONTROL_PUBLICATION` function, it returns the status of the publication specified by the `PUBLICATION-ID`.
17	`SETUSTATUS`	■ Used with the `SYNCPOINT` function to cancel the conversation after committing a UOW. ■ Used with the `CONTROL_PUBLICATION` function, it is used by the publisher to update the optional application-specific user status of the specified publication.
18	`ANY`	Used with the `RECEIVE` function to indicate that the `RECEIVE` will be satisfied by a message that is or is not part of a UOW.
19		No longer used.

Number	Option	Description
20	DURABLE	Used with the SUBSCRIBE function to establish a durable subscription to a topic with attribute ALLOW-DURABLE=YES.
21	CHECKSERVICE	Use with the KERNELVERS function to 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).

	Function																		
ACI Field	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	LOCTRANS	SETSSLPARMS	SEND_PUBLICATION	RECEIVE_PUBLICATION	SUBSCRIBE	UNSUBSCRIBE	CONTROL_PUBLICATION	REPLY_ERROR
ADCOUNT		Rt								Rt				Rt	Rt			Rt	
API-TYPE	Rq	Rq	Rq	Rq	Rq	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	Rq	Rq	Rq	Rq	Rq
BROKER-ID	Rq	Rq	Rq	Rq	Rq	Rq		Rq	Rq	Rq	Rq		Rq	Rq	Rq	Rq	Rq	Rq	Rq
CLIENT-UID		Rt													Rt				
COMMITTIME	Rt	Rt								Rt				Rt	Rt			Rt	
COMPRESSLEVEL	O Rt	O Rt												O Rt	O Rt				O
CONV-ID	Rq	Rq	Rq	Rq						Rq Rt									Rq
CONV-STAT		Rt													Rt				
DATA-ARCH	O	O												O	O				O
ENCRYPTION-LEVEL	O	O			O					O									
ENVIRONMENT	O	O												O	O				O
ERROR-CODE	Rt	Rt	Rt	Rt	Rt	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	O	O	O	O	O
FORCE-LOGON	O	O	O	O	O	O		O	O	O				O	O	O	O	O	O

	Function																		
ACI Field	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	LOCTRANS	SETSSLPARMS	SEND_PUBLICATION	RECEIVE_PUBLICATION	SUBSCRIBE	UNSUBSCRIBE	CONTROL_PUBLICATION	REPLY_ERROR
FUNCTION	Rq	Rq	Rq	Rq	Rq	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	Rq	Rq	Rq	Rq	Rq
LOCALE-STRING	O	O												O	O				O
MSG-ID																			
NEWPASSWORD	O				O			O											
PARTNER-BROKER-ID (deprecated)	O	Rt	Rt	Rt	Rt	Rt		Rt	Rt	Rt	Rt			Rt	Rt	Rt	Rt	Rt	Rt
OPTION	O	O	O	O	O	O			O	O	O	O		O	O	O		Rq	O
PASSWORD	O				O			O											O
PUBLICATION-ID														Rq Rt	Rq			Rq Rt	
RECEIVE-LENGTH	O	Rq					Rq								Rq				
RETURN-LENGTH	Rt	Rt					Rt								Rt				
SECURITY-TOKEN	O	O	O	O	O	O	O	O	O	O				O	O	O	O	O	O
SEND-LENGTH	Rq											Rq	Rq	Rq					Rq
SERVER-CLASS	Rq	O		O	Rq	O				O Rt		Rt							
SERVER-NAME	Rq	O		O	Rq	O				O Rt		Rt							
SERVICE	Rq	O		O	Rq	O				O Rt		Rt							
STORE	O	Rt												Rt	Rt				
TOKEN	O	O	O	O	O	O		O	O					Rq	Rq	Rq	Rq	Rq	O
TOPIC														Rq	Rq	Rq	Rq	Rq	
UOWID	O Rt	O Rt								Rq									
UOWSTATUS	Rt	Rt		Rt						Rt				Rt	Rt				
UOW-STATUS-PERSIST										O									
USER-DATA	O	Rt								Rt									O
USER-ID	Rq	Rq	Rq	Rq	Rq	Rq		Rq	Rq	Rq				Rq	Rq	Rq	Rq	Rq	Rq

	Function																		
ACI Field	SEND	RECEIVE	UNDO	EOC	REGISTER	DEREGISTER	VERSION	LOGON	LOGOFF	SYNCPOINT	KERNELVERS	LOCTRANS	SETSSLPARMS	SEND_PUBLICATION	RECEIVE_PUBLICATION	SUBSCRIBE	UNSUBSCRIBE	CONTROL_PUBLICATION	REPLY_ERROR
USTATUS	O Rt	O Rt		O Rt						O Rt				O Rt	O Rt			O Rt	
UWSTAT-LIFETIME										O									
UWTIME	O																		
WAIT	O	O													O				O

10 Broker UOW Status Transition

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

See also *Broker ACI Fields* in the ACI Programming documentation | *Broker ACI Functions* in the EntireX Broker ACI Programming documentation | *Error Messages and Codes*.

Initial UOW Status: NULL | Received

No.	Initial UOW Status	Action	Resulting UOW Status				Description
			PU&PS	PU&NPS	NPU&PS	NPU&NPS	
2	Received	Send	Received	Received	Received	Received	
3	Received	Commit	Accepted	Accepted	Accepted	Accepted	
4	Received	ReStart	BackedOut	NULL	Discarded	NULL	
5	Received	BackOut	BackedOut	NULL	BackedOut	NULL	
6	Received	TimeOut	BackedOut	NULL	BackedOut	NULL	R6: This action can only be a conversation timeout since a UOW only exists once it is committed.
7	Received	Delete	Received	Received	Received	Received	
8	Received	Cancel	Received	Received	Received	Received	
9	Received	Receive	Received	Received	Received	Received	

Initial UOW Status: Accepted | Delivered

No.	Initial UOW Status	Action	Resulting UOW Status				Description
			PU&PS	PU&NPS	NPU&PS	NPU&NPS	
10	Accepted	Receive	Delivered	Delivered	Delivered	Delivered	
11	Accepted	Timeout	Timedout	NULL	Timedout	NULL	
12	Accepted	Restart	Accepted	Accepted	Discarded	NULL	
13	Accepted	Cancel	Cancelled	NULL	Cancelled	NULL	
14	Accepted	Delete	Accepted	Accepted	Accepted	Accepted	
15	Accepted	BackOut	Accepted	Accepted	Accepted	Accepted	
16	Accepted	Send	Accepted	Accepted	Accepted	Accepted	
17	Accepted	Commit	Accepted	Accepted	Accepted	Accepted	
18	Delivered	Receive	Delivered	Delivered	Delivered	Delivered	
19	Delivered	Commit	Processed	NULL	Processed	NULL	
20	Delivered	Cancel	Cancelled	NULL	Cancelled	NULL	R20: Cancel can only be issued by receiver of the UOW
21	Delivered	BackOut	Accepted	Accepted	Accepted	Accepted	
22	Delivered	TimeOut	Timedout	NULL	NULL	NULL	
23	Delivered	Restart	Accepted	Accepted	Discarded	NULL	
24	Delivered	Delete	Delivered	Delivered	Delivered	Delivered	
26	Delivered	Send	Delivered	Delivered	Delivered	Delivered	

Initial UOW Status: Processed | Timedout

No.	Initial UOW Status	Action	Resulting UOW Status				Description
			PU&PS	PU&NPS	NPU&PS	NPU&NPS	
27	Processed	Delete	NULL	N/A	NULL	N/A	Processed is a STABLE UOW status:
28	Processed	Timeout	NULL	NULL	NULL	N/A	All actions and transitions refer to the status of a UOW.
29	Processed	Restart	Processed	N/A	Processed	N/A	
30	Processed	Backout	Processed	N/A	Processed	N/A	
31	Processed	Cancel	Processed	N/A	Processed	N/A	
32	Processed	Commit	Processed	N/A	Processed	N/A	
33	Processed	Receive	Processed	N/A	Processed	N/A	
34	Processed	Send	Processed	N/A	Processed	N/A	
35	Timedout	Restart	Timeout	N/A	Timeout	N/A	Timedout is a STABLE UOW status:
36	Timedout	Delete	NULL	N/A	NULL	N/A	All actions and transitions refer to the status of a UOW.
37	Timedout	Timeout	NULL	N/A	NULL	N/A	
38	Timedout	Send	Timedout	N/A	Timedout	N/A	
39	Timedout	Receive	Timedout	N/A	Timedout	N/A	
40	Timedout	Commit	Timedout	N/A	Timedout	N/A	
41	Timedout	Backout	Timedout	N/A	Timedout	N/A	
42	Timedout	Cancel	Timedout	N/A	Timedout	N/A	

Initial UOW Status: Cancelled | Discarded | Backedout

No.	Initial UOW Status	Action	Resulting UOW Status				Description
			PU&PS	PU&NPS	NPU&PS	NPU&NPS	
43	Cancelled	Delete	NULL	N/A	NULL	N/A	Cancelled is a STABLE UOW status:
44	Cancelled	Restart	Cancelled	N/A	Cancelled	N/A	All actions and transitions refer to the status of a UOW.
45	Cancelled	TimeOut	NULL	N/A	NULL	N/A	
46	Cancelled	Send	Cancelled	N/A	Cancelled	N/A	
47	Cancelled	Receive	Cancelled	N/A	Cancelled	N/A	
48	Cancelled	Commit	Cancelled	N/A	Cancelled	N/A	
49	Cancelled	Backout	Cancelled	N/A	Cancelled	N/A	
50	Cancelled	Cancel	Cancelled	N/A	Cancelled	N/A	
51	Discarded	Delete	N/A	N/A	NULL	N/A	Discarded is a STABLE UOW status:
52	Discarded	TimeOut	N/A	N/A	NULL	N/A	All actions and transitions refer to the status of a UOW.
53	Discarded	Restart	N/A	N/A	Discarded	N/A	
54	Discarded	Cancel	N/A	N/A	Discarded	N/A	
55	Discarded	Send	N/A	N/A	Discarded	N/A	
56	Discarded	Receive	N/A	N/A	Discarded	N/A	
57	Discarded	Commit	N/A	N/A	Discarded	N/A	
58	Discarded	Backout	N/A	N/A	Discarded	N/A	
59	BackedOut	TimeOut	NULL	N/A	NULL	N/A	BackedOut is a STABLE UOW status:
60	BackedOut	Cancel	BackedOut	N/A	BackedOut	N/A	All actions and transitions refer to the status of a UOW
61	BackedOut	Restart	BackedOut	N/A	BackedOut	N/A	
62	BackedOut	Send	BackedOut	N/A	BackedOut	N/A	
63	BackedOut	Receive	BackedOut	N/A	BackedOut	N/A	
64	BackedOut	Commit	BackedOut	N/A	BackedOut	N/A	
65	BackedOut	Delete	NULL	N/A	NULL	N/A	
66	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

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* under *Broker Command and Information Services*.

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* under *Broker CIS Data Structures* in the ACI Programming documentation.

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 14 SUBSCRIBER [1] 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

Field Name	Format	CIS Interface Version	Comment
			12 PURGE 29 RESET-USER 31 RESUME 19 SET-CMDLOG-FILTER 8 SHUTDOWN 33 START 36 STATUS 32 STOP 15 SUBSCRIBE [(1)] 30 SUSPEND 26 SWITCH-CMDLOG 35 TRACE-FLUSH 2 TRACE-OFF 1 TRACE-ON 34 TRAP-ERROR 16 UNSUBSCRIBE [(1)]
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
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.

Field Name	Format	CIS Interface Version	Comment
TOPIC [1]	A96	4	Selection field. Optional. Specifies the topic to be subscribed or unsubscribed to.
UID	A32	4	Selection field. Optional. Specifies the user name for subscription/unsubscription and participant shutdown.
TOKEN	A32	4	Selection field. Optional. Specifies the token name for subscription/unsubscription and 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.
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 (i.e. client, server, publisher, subscriber) 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.

[1] See *Writing Applications: Publish and Subscribe* in the ACI Programming documentation.

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 TOPIC, 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 TOPIC, 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 TOPIC, 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 TOPIC, 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.)
	TRACE-OFF		Set trace off in the persistent store.

Object Type	Command	Option	Comment
	TRACE-ON	LEVEL	SetTRACE-LEVEL on in the persistent store.
PARTICIPANT	SHUTDOWN	IMMED	Shutdown participant immediately. The participant must be identified, using fields P-USER-ID under *Broker CIS Data Structures* in the ACI Programming documentation, 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.
	TRACE-ON	LEVEL	Set TRACE-LEVEL on in EntireX Security.
SERVICE	SHUTDOWN	IMMED	
		QUIESCE	
		class/server/service	
SUBSCRIBER	SUBSCRIBE		Subscribe a user to a topic.
	UNSUBSCRIBE		Unsubscribe a user to a topic.
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.

Object Type	Command	Option	Comment
	`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* under *Broker CIS Data Structures* in the ACI Programming documentation or *Information Request Structure* under *Broker CIS Data Structures* in the ACI Programming documentation.

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* under *Error Messages and 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.
MAX-TOPIC-LEN	I4	4	Length of longest TOPIC value in total object list. This field is only relevant if the object-specific structure for the object list contains the TOPIC 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*.

Field Name	Format	CIS Interface Version	Comment
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.
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* under *Writing Applications: Command and Information Services* in the ACI documentation.

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
				PSFFILE 13 Info on the B-Tree persistent store (no longer supported) PUBLICATION 16 Info on active publications PUBLISHER 15 Info on active publishers 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) SUBSCRIBER 14 Info on active subscribers TCP 19 Info on TCP communicators. (1) TOPIC 17 Info on active topics 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

Field Name	Format	CIS Interface Version	Opt/ Req	Comment
				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.
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.
TOPIC	A96	4	O	Selection field. Specifies the topic name.
PUBLICATION-ID	A16	4	O	Selection field. Specifies the publication ID.

Field Name	Format	CIS Interface Version	Opt/ Req	Comment
SUBSCRIPTION-TYPE	I2	4	O	Selection field. Specifies the subscription type: 1 DURABLE 2 NON-DURABLE
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
TOPIC	TOPIC
PBN-ID	PUBLICATION-ID
SBN-TYP	SUBSCRIPTION-TYPE

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

Selection Criteria for PUBLISHER Objects (Version 4 and above)

Selection	PUID	UID	TK	TOPIC	PBN-ID
Publisher [(1)]	Y	I	I	I	I
Publisher [(2)]	N	Y	Y	I	I
Publishers with UID	N	Y	N	I	I
Publishers with TK	N	N	Y	I	I
All publishers	N	N	N	I	I

Selection Criteria for PUBLICATION (Version 4 and above)

Selection	PUID	UID	TK	TOPIC	PBN-ID
Publications of this publisher/subscriber [(1)]	Y	I	I	I	I
Publications of this publisher/subscriber [(2)]	N	Y	Y	I	I
Publications of this topic	N	N	N	Y	I
The publications with PBN-ID	N	N	N	N	Y
All publications	N	N	N	N	N

Selection Criteria for TOPIC (Version 4 and above)

Selection	PUID	UID	TK	TOPIC	PBN-ID
Topics subscribed to by this subscriber [(1)]	Y	I	I	I	I
Topics subscribed to by this subscriber [(2)]	N	Y	Y	I	I
Topics with this TOPIC	N	N	N	Y	I
All topics	N	N	N	N	N

Selection Criteria for SUBSCRIBER (Version 4 and above)

Selection	PUID	UID	TK	TOPIC	SBN-TYP
Subscriber [(1)]	Y	I	I	I	I
Subscriber [(2)]	N	Y	Y	I	I
Subscribers with UID	N	Y	N	I	I
Subscribers with TK	N	N	Y	I	I
All subscribers with SBN-TYP	N	N	N	I	Y
All subscribers with TOPIC	N	N	N	Y	I

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)
- PSFFILE-OBJECT (Struct (INFO_PSFFILE)
- PUBLICATION-OBJECT (Struct INFO_PUBLICATION)
- PUBLISHER-OBJECT (Struct INFO_PUBLISHER)
- 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)
- SUBSCRIBER-OBJECT (Struct INFO_SUBSCRIBER)
- TCP-OBJECT (Struct INFO_TCP)
- TOPIC-OBJECT (Struct INFO_TOPIC)
- 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.
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.
LUWTIME	I4	2	Maximum UOW lifetime
LMAXDELCNT	I4	2	Currently not in use. (Count is always zero.)

Field Name	Format	CIS Interface Version	Description / Action
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
SNMP-LICENSED	I1	3	SNMP licensed: 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)
PUB-SUB	I1	4	Publish and subscribe: 0 NO 1 YES
HIGHEST-API-VERSION	I1	4	For example: 0x06.
HIGHEST-CIS-VERSION	I1	4	For example: 0x06.
PSTORE-CONNECTED	I1	4	0 NO 1 YES
NUM-TOPIC	I4	4	Number of topics defined (see NUM-TOPIC).
TOPIC-ACT	I4	4	Number of topics active.
NUM-SUBSCRIBER	I4	4	Number of subscribers defined (see NUM-SUBSCRIBER).
SUBSCRIBER-ACT	I4	4	Number of subscribers active.
SUBSCRIBER-HIGH	I4	4	Highest number of subscribers active since Broker started.
NUM-PUBLISHER	I4	4	Number of publishers defined (see NUM-PUBLISHER).
PUBLISHER-ACT	I4	4	Number of publishers active.
PUBLISHER-HIGH	I4	4	Highest number of publishers active since Broker started.
NUM-PUBLICATION	I4	4	Number of publications defined (see NUM-PUBLICATION).
PUBLICATION-HIGH	I4	4	Highest number of publications active since Broker started.
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.

Field Name	Format	CIS Interface Version	Description / Action
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-SUBSCB	I4	5	Number of free SUBSCB entries (subscriber control block).
NUM-FREE-TCB	I4	5	Number of free TCB entries (topic control block).
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).
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

Field Name	Format	CIS Interface Version	Description / Action
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.
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

Field Name	Format	CIS Interface Version	Description / Action
			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.
CTOPIC-UPDATES	I1	7	Topic updates: 0 NO 1 YES See broker attribute TOPIC-UPDATES.
CTRANSPORT-NET	I1	7	Was TRANSPORT=NET specified? 0 NO 1 YES See broker attribute TRANSPORT=NET.
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.

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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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 under *Broker CIS Data Structures* in the ACI Programming documentation 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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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.

Field Name	Format	CIS Interface Version	Description / Action
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 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.

Field Name	Format	CIS Interface Version	Description / Action
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.
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.
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.

Field Name	Format	CIS Interface Version	Description / Action
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 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/IP section of Broker attribute file.

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.
TOPIC	A96	5	Topic.
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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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.

Field Name	Format	CIS Interface Version	Description / Action
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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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)
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

Field Name	Format	CIS Interface Version	Description / Action
			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.

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.
EXT-SUBSCRIBER-NUM-POOLS	I4	7	PSTORE-SUBSCRIBER: Number of pools.
EXT-SUBSCRIBER-SIZE-ALL-POOLS	I4	7	PSTORE-SUBSCRIBER: Size of all pools in bytes.
EXT-SUBSCRIBER-SIZE-ONE-POOL	I4	7	PSTORE-SUBSCRIBER: Size of one pool in bytes.
EXT-TOPIC-NUM-POOLS	I4	7	PSTORE-TOPIC: Number of pools.
EXT-TOPIC-SIZE-ALL-POOLS	I4	7	PSTORE-TOPIC: Size of all pools in bytes.
EXT-TOPIC-SIZE-ONE-POOL	I4	7	PSTORE-TOPIC: 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.

Field Name	Format	CIS Interface Version	Description / Action
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.
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.
SUBSCRIPTION-NUM-POOLS	I4	7	SUBSCRIPTION: Number of pools.
SUBSCRIPTION-SIZE-ALL-POOLS	I4	7	SUBSCRIPTION: Size of all pools in bytes.
SUBSCRIPTION-SIZE-ONE-POOL	I4	7	SUBSCRIPTION: 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.
TOPIC-ATTRIBUTE-NUM-POOLS	I4	7	TOPIC-ATTRIBUTE: Number of pools.
TOPIC-ATTRIBUTE-SIZE-ALL-POOLS	I4	7	TOPIC-ATTRIBUTE: Size of all pools in bytes.
TOPIC-ATTRIBUTE-SIZE-ONE-POOL	I4	7	TOPIC-ATTRIBUTE: Size of one pool in bytes.
TOPIC-NUM-POOLS	I4	7	TOPIC: Number of pools.
TOPIC-SIZE-ALL-POOLS	I4	7	TOPIC: Size of all pools in bytes.
TOPIC-SIZE-ONE-POOL	I4	7	TOPIC: Size of one pool in bytes.
TOPIC-EXT-NUM-POOLS	I4	7	TOPIC-EXT: Number of pools.
TOPIC-EXT-SIZE-ALL-POOLS	I4	7	TOPIC-EXT: Size of all pools in bytes.
TOPIC-EXT-SIZE-ONE-POOL	I4	7	TOPIC-EXT: 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
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

Field Name	Format	CIS Interface Version	Description / Action
			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	Time UOW created.
UWLIFETIME	I4	2	UOW lifetime.

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.

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.
DIV-SH-FORMAT-TOD	A16	2	TOD of persistent store last format in YYYYMMDDHHMMSST.
DIV-SH-FORMAT-VERS	I4	2	Persistent store format version.
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.
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	I4	2	Size of array/hash modulus.

Field Name	Format	CIS Interface Version	Description / Action
DIV-SH-CP-DEF-CNT	I4	2	Number of cell pool definitions.
DIV-CP-NAME	A8	2	Cell pool name.
DIV-CP-CELL-SIZE	I4	2	Cell size.
DIV-CP-CELL-TOTAL	I4	2	Total number of cells.
DIV-CP-CELL-AVAIL	I4	2	Number of cells available.
DIV-CP-EXTENT-CNT	I4	2	Number of cell pool extents.
DIV-CP-QUERY-RC	I4	2	Return code from cell pool query.
DIV-CX-STATUS	I4	2	Cell pool extent status.
DIV-CX-EXTENT-ADDR	B4	2	Address of cell pool extent.
DIV-CX-EXTENT-LEN	I4	2	Length of cell pool extent.
DIV-CX-AREA-ADDR	B4	2	Address of cell area.
DIV-CX-AREA-LEN	I4	2	Length of cell area.
DIV-CX-CELL-TOTAL	I4	2	Number of cells in extent.
DIV-CX-CELL-AVAIL	I4	2	Number of cells available in extent.
DIV-CX-QUERY-RC	I4	2	Return code from cell pool extent query.
TRACE-LEVEL	I2	5	PSTORE trace level.
NOTUSED0	I2	5	Alignment.

PSFFILE-OBJECT (Struct (INFO_PSFFILE)

Field Name	Format	CIS Interface Version	Description / Action
FORMAT-VERSION	I4	4	Store format version.
COLD-START-TOD	A16	4	Store cold-start time.
HOT-STARTS	I4	4	Number of hot starts since format.
SMSGINDEXDAT	A256	4	Message index data file name.
LMSGINDEXDAT	I4	4	Message index data file size.
SMSGINDEXNDX	A256	4	Message index index file name.
LMSGINDEXNDX	I4	4	Message index index file size.
SSTATUSDAT	A256	4	Status data file name.
LSTATUSDAT	I4	4	Status data file size.
SSTATUSNDX	A256	4	Status index file name.
LSTATUSNDX	I4	4	Status index file size.
SMSGDAT	A256	4	Message data file name.
LMSGDAT	I4	4	Message data file size.

PUBLICATION-OBJECT (Struct INFO_PUBLICATION)

Field Name	Format	CIS Interface Version	Description / Action
PUBLICATION-ID	A16	4	Publication ID.
PUBLISHER-USER-ID	A32	4	Publisher's user ID.
PUBLISHER-P-USER-ID	A28	4	Publisher's physical user ID.
PUBLISHER-P-USER-ID-CHAR	A28	4	No longer used.
TOKEN	A32	4	Publisher's token.
TOPIC	A96	4	Topic name.
LAST-ACTIVE	I4	4	Time (in seconds) since last activity.
EXPIRATION-TIME	I4	4	Expiration time. Number of seconds since 1 JAN 1970.
LMSQCNT	I4	4	Number of messages.
STATUS	I1	4	Publication status: 1 RECEIVED 2 ACCEPTED 3 DELIVERED 4 BACKEDOUT

Field Name	Format	CIS Interface Version	Description / Action
			5 PROCESSED 6 CANCELLED 7 TIMEOUT 8 DISCARDED
UNUSED1	A3	4	Unused.

PUBLISHER-OBJECT (Struct INFO_PUBLISHER)

Field Name	Format	CIS Interface Version	Description / Action
USER-ID	A32	4	User ID.
P-USER-ID	A28	4	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	4	No longer used.
TOKEN	A32	4	Token.
CHAR-SET	I2	4	Publisher's character set.
ENDIAN	I2	4	Publisher's endianness.
STATUS	I2	4	Publisher's status.
UNUSED1	I2	4	Unused.
WAIT-PUBL-TYPE	A16	4	Publication that publisher is waiting for.
WAIT-TOPIC	A96	4	Topic that publisher is waiting for.
PUBL-ACT	I4	4	Number of active publications.
TOPIC-ACT	I4	4	Number of topics offered.
LAST-ACTIVITY	I4	4	Time (in seconds) since last activity.
NONACT	I4	4	Non activity timeout.
WAIT-NEW	I4	4	Accumulated time (in seconds) of waits for new publications.
NUM-WAIT-NEW	I4	4	Number of waits for new publications.
WAIT-OLD	I4	4	Accumulated time (in seconds) of waits for old publications.
NUM-WAIT-OLD	I4	4	Number of waits for old publications.
SUM-PUBL	I4	4	Total number of publications.
IP-ADDRESS	A16	4	IPv4 address of publisher.
HOST-NAME	A256	4	Host name of publisher.
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

Field Name	Format	CIS Interface Version	Description / Action
			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_5	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.
SEQNO	I4	7	Unique sequence number of publisher. 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 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/IP section of Broker attribute file.

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.
EXT-SUBSCRIBER-ENTRIES-ALLOCATED	I4	7	PSTORE-SUBSCRIBER: Number of entries allocated.
EXT-SUBSCRIBER-ENTRIES-FREE	I4	7	PSTORE-SUBSCRIBER: Number of entries free.
EXT-SUBSCRIBER-ENTRIES-USED	I4	7	PSTORE-SUBSCRIBER: Number of entries used.

Field Name	Format	CIS Interface Version	Description / Action
EXT-TOPIC-ENTRIES-ALLOCATED	I4	7	PSTORE-TOPIC: Number of entries allocated.
EXT-TOPIC-ENTRIES-FREE	I4	7	PSTORE-TOPIC: Number of entries free.
EXT-TOPIC-ENTRIES-USED	I4	7	PSTORE-TOPIC: 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.
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.

Field Name	Format	CIS Interface Version	Description / Action
SERVICE-EXT-ENTRIES-USED	I4	7	SERVICE-EXT: Number of entries used.
SUBSCRIPTION-ENTRIES-ALLOCATED	I4	7	SUBSCRIPTION: Number of entries allocated.
SUBSCRIPTION-ENTRIES-FREE	I4	7	SUBSCRIPTION: Number of entries free.
SUBSCRIPTION-ENTRIES-USED	I4	7	SUBSCRIPTION: 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.
TOPIC-ATTRIBUTE-ENTRIES-ALLOCATED	I4	7	TOPIC-ATTRIBUTE: Number of entries allocated.
TOPIC-ATTRIBUTE-ENTRIES-FREE	I4	7	TOPIC-ATTRIBUTE: Number of entries free.
TOPIC-ATTRIBUTE-ENTRIES-USED	I4	7	TOPIC-ATTRIBUTE: Number of entries used.
TOPIC-ENTRIES-ALLOCATED	I4	7	TOPIC: Number of entries allocated.
TOPIC-ENTRIES-FREE	I4	7	TOPIC: Number of entries free.
TOPIC-ENTRIES-USED	I4	7	TOPIC: Number of entries used.
TOPIC-EXT-ENTRIES-ALLOCATED	I4	7	TOPIC-EXT: Number of entries allocated.
TOPIC-EXT-ENTRIES-FREE	I4	7	TOPIC-EXT: Number of entries free.
TOPIC-EXT-ENTRIES-USED	I4	7	TOPIC-EXT: 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.
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* under *Broker CIS Data Structures* in the ACI Programming documentation.

Field Name	Format	CIS Interface Version	Description / Action
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.
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
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	Encryption level
ATTACH-MGRS-ACT	I4	4	Number of attach servers active
LUWSTAT-ADD-TIME	I4	4	Unit of work status additional lifetime

Field Name	Format	CIS Interface Version	Description / Action
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.

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.
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.
NUM-TOPIC	I4	7	Number of topics defined (see NUM-TOPIC).
TOPIC-ACT	I4	7	Number of topics active.
NUM-PUBLISHER	I4	7	Number of publishers defined (see NUM-PUBLISHER).
PUBLISHER-ACT	I4	7	Number of publishers active.
PUBLISHER-HIGH	I4	7	Highest number of publishers active since Broker started.
NUM-SUBSCRIBER	I4	7	Number of subscribers defined (see NUM-SUBSCRIBER).
SUBSCRIBER-ACT	I4	7	Number of subscribers active.
SUBSCRIBER-HIGH	I4	7	Highest number of subscribers active since Broker started.
NUM-PUBLICATION	I4	7	Number of publications defined (see NUM-PUBLICATION).
PUBLICATION-ACT	I4	7	Number of publications active.
PUBLICATION-HIGH	I4	7	Highest number of publications active since Broker started.

SUBSCRIBER-OBJECT (Struct INFO_SUBSCRIBER)

Field Name	Format	CIS Interface Version	Description / Action
TOPIC	A96	4	Topic name.
USER-ID	A32	4	User ID.
P-USER-ID	A28	4	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	4	No longer used.
TOKEN	A32	4	Token.
SUBSCRIPTION-TIME	I4	4	Subscription time.
LAST-ACTIVITY-TIME	I4	4	Last activity time.
EXPIRATION-TIME	I4	4	Expiration time.
LAST-COMMITTED	A16	4	Last committed publication.
LAST-RECEIVED	A16	4	Last received publication.
DURABLE	I1	4	Durable flag: 0 NO

Field Name	Format	CIS Interface Version	Description / Action
			1 YES
SWAPPED-OUT	I1	4	Durable subscriber swapped out? 0 NO 1 YES
UNUSED1	I2	4	Unused.
CHAR-SET	I2	5	User's character set: 34 EBCDIC_IBM 66 EBCDIC_SNI 1 ASCII_PC_386 16 ASCII_PC_OS2 128 ASCII_8859_1
BIG-ENDIAN	I2	5	High order first: 0 NO 1 YES
STATUS	I2	5	User's status: 0 Not waiting 5 Waiting
WAIT-PUBLID	A16	5	PUBLID user is waiting for: "NEW" "OLD" "ANY" publication ID.
NOTUSED1	I2	5	Alignment.
NUM-PUBLICATIONS-ACTIVE	I4	5	Number of active publications.
NONACT-TIME	I4	5	Nonactivity timeout.
WAIT-NEW	I4	5	Accumulated time of waits for new publications.
NUM-WAIT-NEW	I4	5	Number of waits for new publ.
WAIT-OLD	I4	5	Accumulated time of waits for messages from publisher.
NUM-WAIT-OLD	I4	5	Number of waits for publshr msgs.
IP-ADDRESS	A16	5	IPv4 address of subscriber.
HOST-NAME	A256	5	Host name of subscriber.
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

Field Name	Format	CIS Interface Version	Description / Action
			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_4`	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.
`SEQNO`	I4	7	Unique sequence number of subscriber. 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 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/IP` section of Broker attribute file.

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* in the general administration documentation.
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.

TOPIC-OBJECT (Struct INFO_TOPIC)

Field Name	Format	CIS Interface Version	Description / Action
TOPIC	A96	4	Topic name.
TRANS	A8	4	Name of translation routine.
CONV	A8	4	Name of conversion routine.
PUBLISHER-NONACT	I4	4	Publisher non-activity timeout.
SUBSCRIBER-NONACT	I4	4	Subscriber non-activity timeout.
SUBSCRIPTION-EXP	I4	4	Subscription expiration time.
PUBLICATION-ACT	I4	4	Number of active publications.
PUBLICATION-HIGH	I4	4	Highest number of publications active since Broker started.
DURABLE-ACT	I4	4	Number of durable subscribers active.
NON-DURABLE-ACT	I4	4	Number of non-durable subscribers active.
LONG-ACT	I4	4	Number of long buffers active.
LONG-HIGH	I4	4	Highest number of long buffers active since Broker started.
SHORT-ACT	I4	4	Number of short buffers active.

Field Name	Format	CIS Interface Version	Description / Action
SHORT-HIGH	I4	4	Highest number of short buffers active since Broker started.
ALLOW-DURABLE	I1	4	Durable subscriber flag: 0 NO 1 YES
ALLOW-SUBSCRIBE	I1	4	Allow SUBSCRIBE request? 0 NO 1 YES
AUTO-COMMIT	I1	4	Auto commit: 0 NO 1 YES
UNUSED1	I1	4	Unused.
NUM-PUBLICATION	I4	5	Number of publications.
NUM-SUBSCRIBER	I4	5	Number of subscribers.
NUM-LONG-MSG-BUFFER	I4	5	Number of long message buffers.
NUM-SHORT-MSG-BUFFER	I4	5	Number of short message buffers.
CONVERSION-PARMS	A255	5	Conversion parameters.
NOTUSED1	I2	5	Alignment.
MAX-PUBLICATION-MSG-SIZE	I4	5	Max publication message size.
RESERVED	I4	5	Reserved for future use.

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

Field Name	Format	CIS Interface Version	Description / Action
IS-SERVER	I1	7	Is user a server? 0 NO 1 YES
IS-PUBLISHER	I1	7	Is user a publisher? 0 NO 1 YES
IS-SUBSCRIBER	I1	7	Is user a subscriber? 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
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* under *Broker CIS Data Structures* in the ACI Programming documentation.
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.
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.

Field Name	Format	CIS Interface Version	Description / Action
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.