software AG

webMethods EntireX

Broker

Version 9.12

October 2016

WEBMETHODS

This document applies to webMethods EntireX Version 9.12 and all subsequent releases.

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

Document ID: EXX-BROKER-912-20181116

Table of Contents

1 About this Documentation

Document Conventions

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

Online Information and Support

Software AG Documentation Website

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

Software AG Empower Product Support Website

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

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

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

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

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

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

Software AG TECHcommunity

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

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

Data Protection

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

I Concepts and Facilities of EntireX Broker

EntireX Broker is a middleware infrastructure that allows application components in a distributed processing environment to communicate with each other. EntireX Broker provides access through the *client and server* communication model. Message queues are employed to provide verifiable delivery of message data in asynchronous communication.

Additionally, EntireX Broker allows each application component to use a different programming interface. As a result, your application components can achieve highly flexible interoperability in a loosely coupled way. EntireX Broker can be used where your application components are located on distributed machines and where different operating systems and TP monitors are used on each machine.

Concept of Interoperability	Introduces the basic concept of EntireX Broker: achieving highly flexible interoperability of distributed application components.
Common Use Cases	Provides specific examples of how your organization can achieve flexible interoperability in a distributed processing environment.
General Architecture of EntireX Broker	Describes the components and transport mechanisms of EntireX Broker within the context of EntireX.
Functionality of EntireX Broker	Provides a brief overview of the functionality provided by EntireX Broker.
Broker Quick Reference	Quick Reference to Broker features and functions.

2 Concept of Interoperability

Note: After viewing this chapter, see the chapter *Common Use Cases*, which supplies specific business examples of the interoperability available through EntireX Broker.

Interoperability and EntireX Broker

This section introduces the basic concept of EntireX Broker: achieving highly flexible interoperability of application components in a distributed processing environment. This concept is described from the perspectives of

- a messaging model
- communication models
- application programming interfaces
- EntireX components

in order to give you a comprehensive, high-level view of how EntireX Broker enables flexible interoperability between distributed application components.

Messaging Model and Interoperability

Introduction

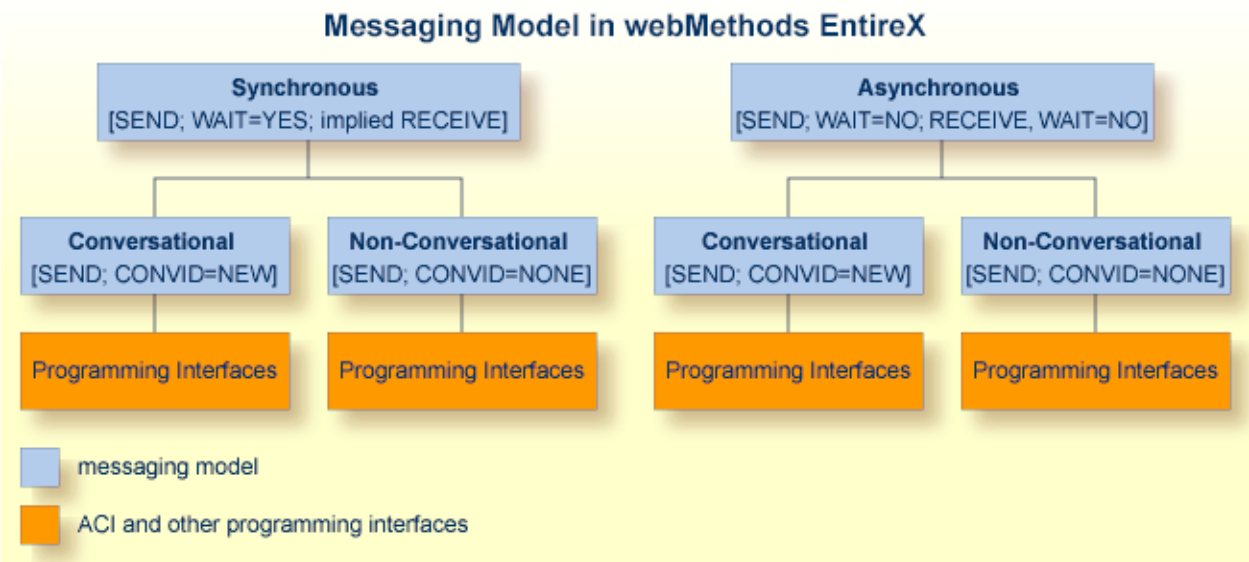
In a distributed processing environment that uses EntireX Broker, communication occurs through application components exchanging messages. An application component offering a service registers it with EntireX Broker (see `REGISTER`); this makes the service available to other application components able to communicate with EntireX Broker. An application component intending to access a service issues its request through EntireX Broker, which then routes the request to the specific application component offering the service.

The following concepts help describe how message exchange is structured in EntireX Broker:

- **Synchronicity**
  The application initiating the request either waits for the result to return, whereby it suspends all processing (synchronous); or it does not wait for the result to return, whereby it is freed to do other processing (asynchronous).
- **Conversationality**
  The request can either be a single pair of messages comprising request/reply (non-conversational); or it can be a sequence of multiple messages which are all part of the same request (conversational).

Overview Diagram

The following diagram shows the two major concepts of EntireX Broker's messaging model: synchronicity and conversationality. See *ACI Syntax of Messaging Model* below for a description of the messaging syntax.



ACI Syntax of Messaging Model

The table below describes the messaging terms mentioned in the diagram above from the viewpoint of the application component initiating the request, as expressed in ACI syntax.

The ACI (Advanced Communication Interface) is the lowest level application programming interface that interacts with EntireX Broker. The ACI is common to all of the messaging models and communication models (see *Communication Models and Interoperability*) of EntireX.

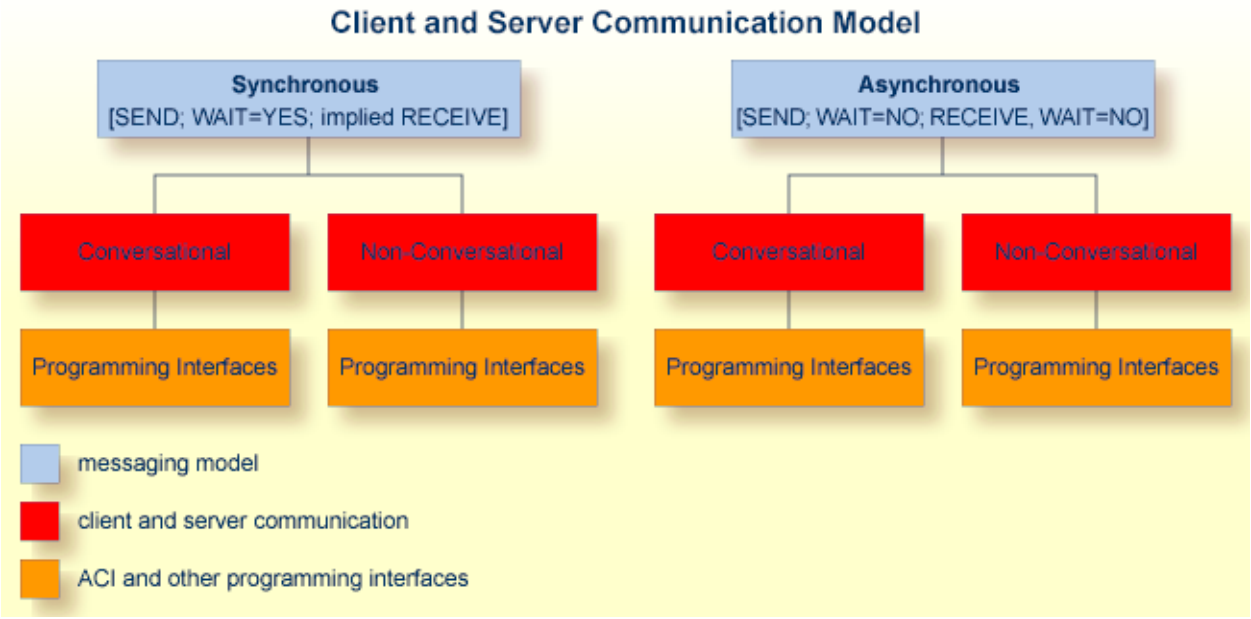
Messaging Term		Client	Server
SYNCHRONICITY	Synchronous	■ SEND [1] ■ WAIT=YES [1]	■ RECEIVE ■ WAIT=YES
	Asynchronous [2]	■ SEND ■ WAIT=NO ■ WAIT=YES	■ RECEIVE ■ WAIT=NO
CONVERSATIONALITY	Conversational [2]	■ SEND ■ CONV-ID=NEW	■ RECEIVE
	Non-conversational [2]	■ SEND ■ CONV-ID=NONE	■ RECEIVE

Notes:

1. The synchronous `SEND, WAIT=YES` command contains an implied `RECEIVE` command.
2. Persistence available. See *Concepts of Persistent Messaging*.

Communication Models and Interoperability

The EntireX Broker uses the communication model client-and-server. This model is based on the connection between exactly two partners: client and server. This model covers the requirements of conversational communication and asynchronous processing.


Client and Server Communication Model
Synchronous
[SEND; WAIT=YES; implied RECEIVE]
Asynchronous
[SEND; WAIT=NO; RECEIVE, WAIT=NO]
Conversational
Non-Conversational
Conversational
Non-Conversational
Programming Interfaces
Programming Interfaces
Programming Interfaces
Programming Interfaces
messaging model
client and server communication
ACI and other programming interfaces

3 Common Use Cases

Introduction

This section provides common use cases of the basic concept of EntireX - achieving highly flexible interoperability of distributed application components. Each use case contains a

- business scenario
- table of interoperability, listing the major components selected for the use case
- diagram of the type of message flow resulting from the combination of these specific components
- stepped table describing the message flow depicted in the diagram.

The common use cases based on the EntireX components Broker and Developer's Kit are provided to show the extent and limitations of the EntireX Broker.

The Developer's Kit contains a set of interfaces for using applications written in various programming languages with EntireX Broker. Developer's Kit enables application components to be "wrapped", i.e. encapsulated, thereby allowing them to behave like an object and be plugged-and-played as needed.

The ACI forms the layer upon which the various wrappers of the Developer's Kit logically exist. This allows application programs to directly utilize the following industry-standard APIs that are exposed through the Developer's Kit and EntireX Broker.

The common use cases in the table below are specific examples of how EntireX Broker provides highly flexible interoperability of application components in a distributed processing environment.

Case	Client	Server	Typical Use
Case 1	ACI	ACI	To integrate applications on separate platforms. (Persistent messaging is described.)
Case 2	JACI	ACI	To integrate applications on separate platforms, whereby the client's application interface is a subset of the ACI.
Case 3	ACI (via Web server)	ACI	To enable Web access to mainframe systems.
Case 4	RPC	RPC	To enable a UNIX or Windows application to access a Natural RPC program.

Case 1: ACI and ACI (including Units of Work)

This case is typically used to integrate applications on separate platforms.

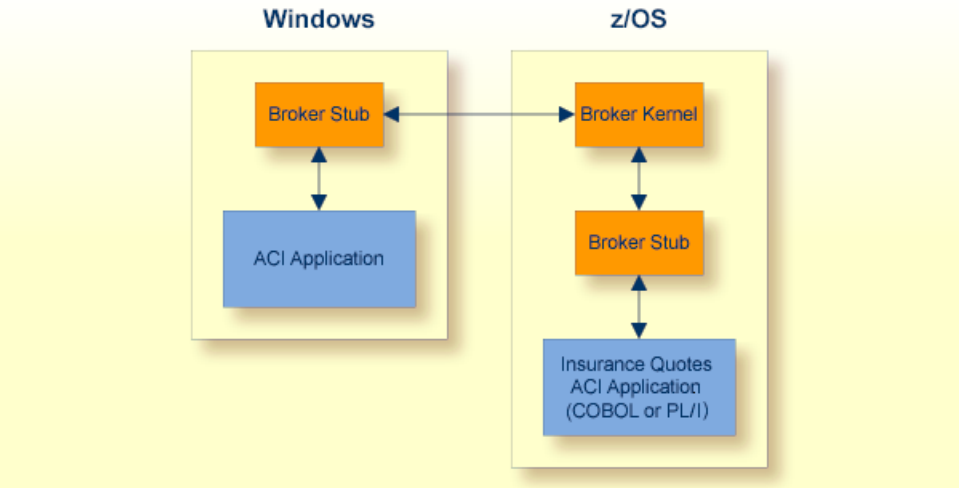
Business Scenario

An insurance company sells its own products as well as those of other insurers. It is company policy for its sales agents to give the most competitive insurance quotes possible to customers. The front-ends used by the sales agent are provided with GUI applications on Windows. To obtain insurance quotes from the back-end data as well as to update those data, the insurance agents must communicate information from/to various mainframe applications written in COBOL and PL/I.

Table of Interoperability

Application Component	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	ACI	Broker	Windows	Visual C	■ Synchronous or asynchronous ■ Conversational or non-conversational
Server	ACI		z/OS	COBOL, PL/I	

Message Flow: ACI and ACI



Description of Steps in Message Flow

1. a. **Synchronous**
   The client program creates a request for information from a mainframe back-end and issues a call via the Broker stub to EntireX Broker.

   With conversational communication, a series of linked requests can be issued, allowing both the client and server to retain context between commands.

   b. **Asynchronous**

   - The client program wants to communicate updated information to the back-end system. It formulates one or more messages within a unit of work (UOW) and performs an asynchronous `SEND` from the stub to the broker.
   - The Broker writes the UOW to the persistent store, enabling the client program to know that the UOW will be processed.

2. a. **Synchronous**
   The server application issues an ACI call via the Broker stub in order to obtain the request from the client program.

   b. **Asynchronous**
   The server application issues a `RECEIVE` command, now or at a later time, in order to obtain the messages from the client program.

3. a. **Synchronous**
   The server application processes the request and returns a message to EntireX Broker via the Broker stub.

b. **Asynchronous**
The server program performs processing to update the data on the back-end system and, only afterwards does it acknowledge that the message has been processed.

4. a. **Synchronous**
The client program receives the reply to the ACI call, allowing the request to be satisfied.

b. **Asynchronous**
The client program can query the status of its messages by `UOWID` in order to determine the status of the back-end processing.

Case 2: JACI and ACI

This case is typically used to integrate applications on separate platforms.

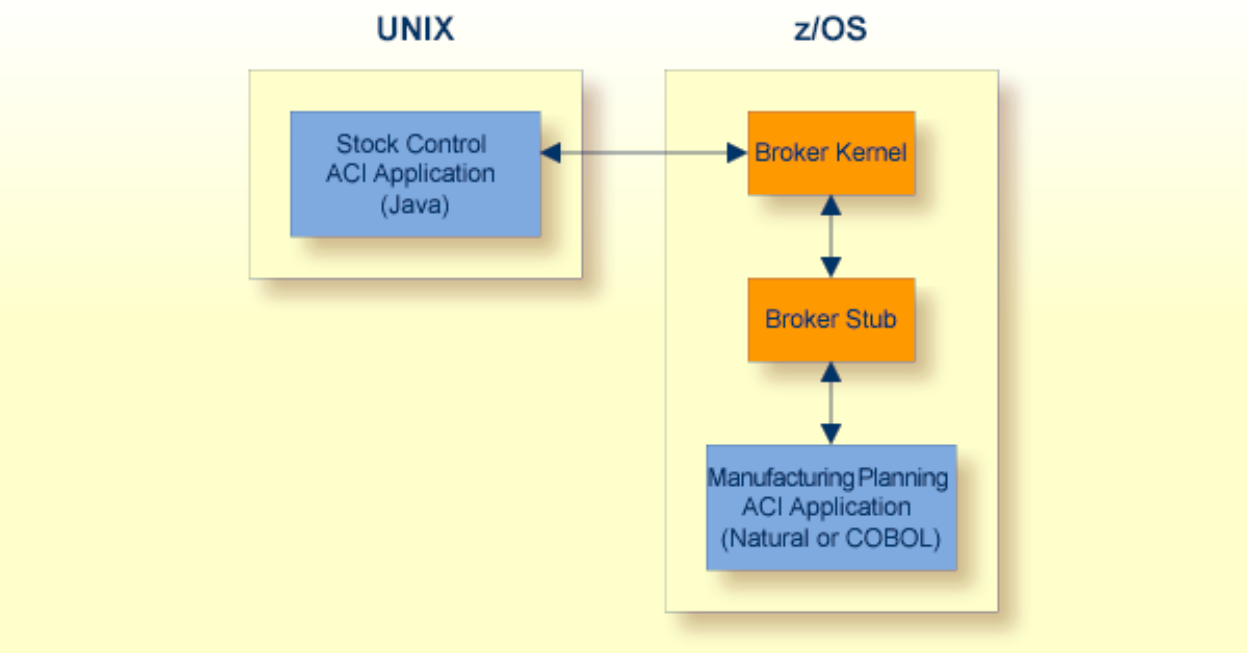
Business Scenario

An organization wants to integrate a UNIX-based stock control system with its existing mainframe-based manufacturing planning systems.

Table of Interoperability

Architecture	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	JACI	Broker	UNIX	Java	■ Synchronous ■ Conversational or Non-conversational
Server	ACI		z/OS	Natural	

Message Flow: JACI and ACI



Description of Steps in Message Flow

1. The client program creates a request and issues a JACI call to EntireX Broker.
2. The server application issues an ACI call via the Broker stub in order to obtain the request from the client program.
3. The server application processes the request and returns a message to EntireX Broker via the Broker stub.
4. The client program receives the reply to the ACI call, allowing the request to be satisfied.

Case 3: ACI (via Web Server) and ACI

This case is typically used to enable Web access to mainframe systems.

Business Scenario

A brokerage has an application which processes orders of personal customers to buy and sell securities. All incoming orders are executed on a back-end system, and some orders are executed at a later time. The incoming orders are in the form of internet communication.

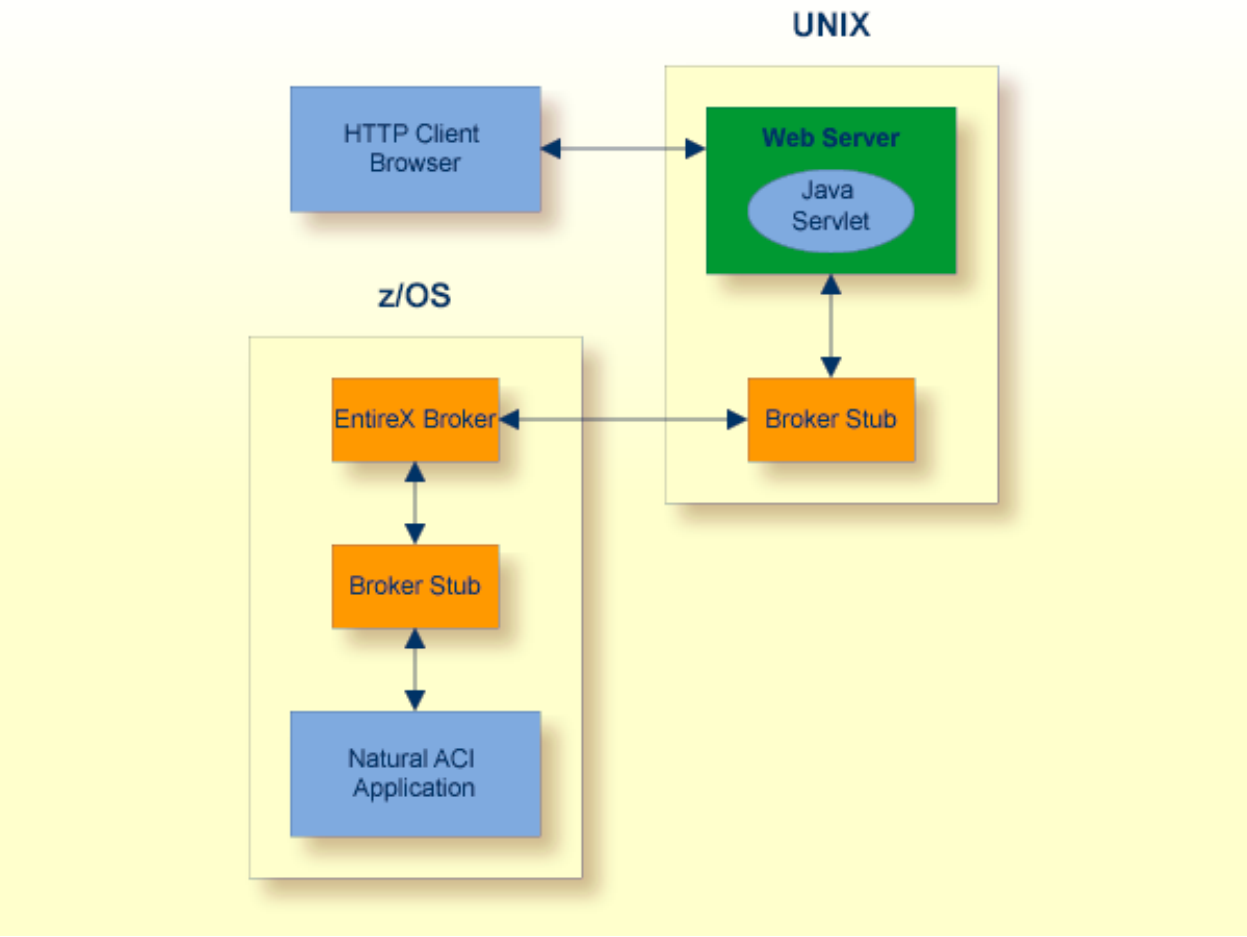
IT Environment

The brokerage uses a Web server as the point-of-entry for incoming orders. These orders are executed either synchronously or asynchronously on a separate back-end system. Located on the brokerage's Web server is an application which is a client to EntireX, which functions as a proxy and provides information to the brokerage's EIS (Enterprise Information System). Because of the critical nature of the orders, units of work are employed to guarantee delivery of the incoming information to the back-end system. This system is robust and can be restarted after failure without loss of data.

Table of Interoperability

Architecture	Programming Interface	EntireX Component	Operating System	Language	Messaging Model
Client	JACI	Broker	UNIX	Java Servlet	■ Synchronous or Asynchronous ■ Conversational
Server	ACI		z/OS	Natural	

Message Flow: ACI and IBM® MQ



Description of Steps in Message Flow

1. The Web browser sends an HTTP request to the Web server.
2. The Web server instantiates a Web page containing the script (ASP).
3. The script creates a request and issues an ACI call via the Broker stub to EntireX Broker.
4. The back-end application issues an ACI call via the Broker stub in order to obtain the request from the script.
5. The back-end application processes the request and returns a message to EntireX Broker via the Broker stub.
6. The script receives the reply to the ACI call, allowing the execution of the Web page to be completed.
7. The Web server returns the information to the Web browser via HTTP, where the Web page is displayed.

Case 4: RPC Wrapper and RPC

This case is typically used to enable a UNIX or Windows application to access a Natural RPC program.

Note: This use case is the most common within EntireX; it employs the EntireX Broker together with the Developer's Kit.

Business Scenario

An organization actively using Software AG technology - including Adabas and Natural - wants to expand use of Software AG technology in order to build new applications accessible to clients executing under UNIX or Windows. To achieve this, the organization runs a client written to use RPC, which makes calls to EntireX Broker. The client, which is written in either Natural, Java or a 3GL language, will invoke any of these three variants:

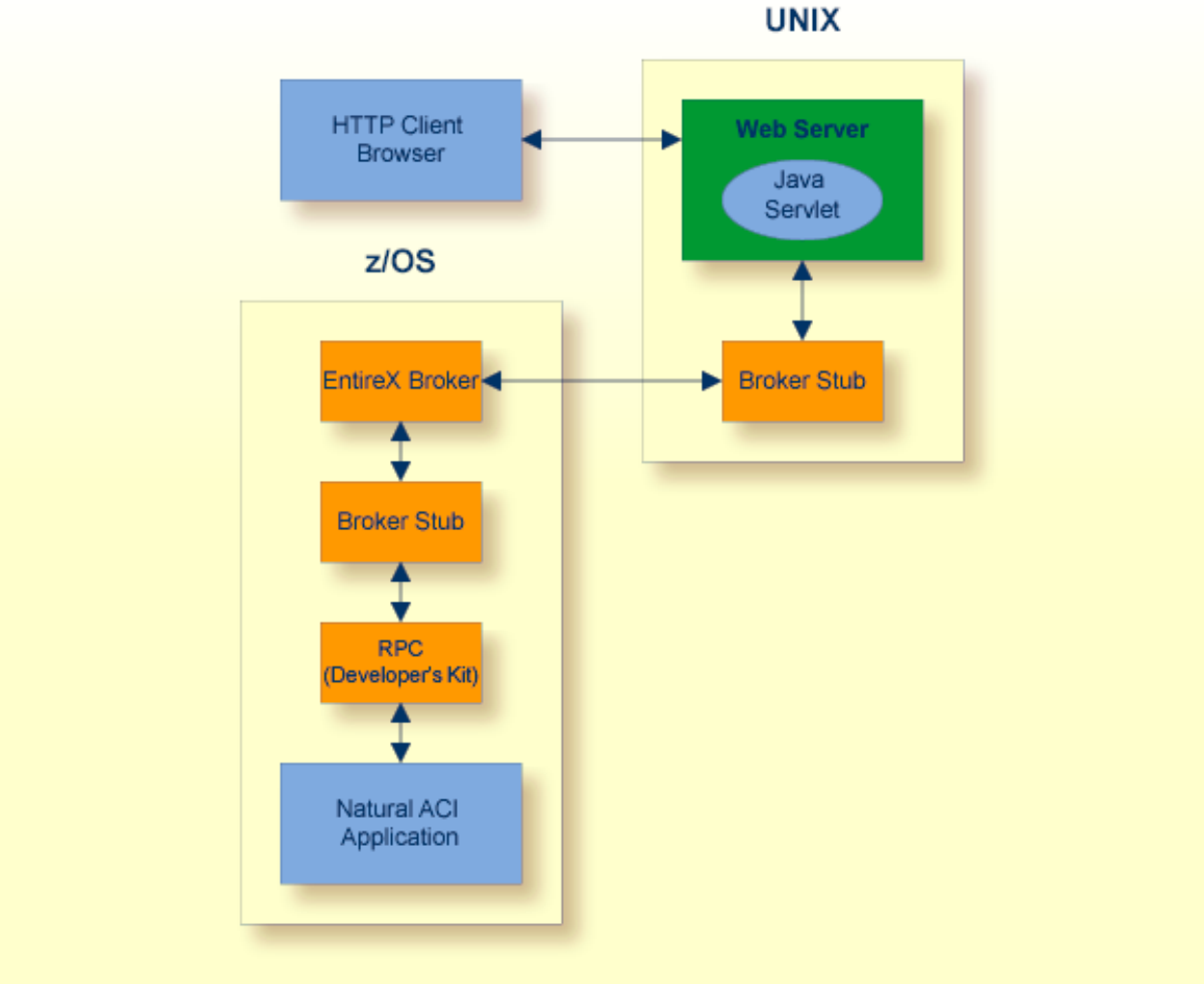
- **(A)**
  RPC programs written in Natural and executing under Natural on z/OS (RPC is available through Natural on z/OS);
- **(B)**
  RPC programs written in Java and executing under the Java RPC Server on UNIX;
- **(C)**
  3GL RPC programs executing under the C RPC Server on Windows.

Table of Interoperability

Application Component		Programming Interface	EntireX Component	Operating System	Language *	Messaging Model
(A)	Client	RPC	EntireX Broker and Developer's Kit	C	Visual Basic	■ Synchronous ■ Conversational or Non-conversational
	Server	RPC		z/OS	Natural	
(B)	Client	RPC		Windows	Natural	
	Server	RPC		UNIX	Java	
(C)	Client	RPC		UNIX	Java	
	Server	RPC		Windows	C (=3GL)	

Message Flow: RPC Wrapper and RPC

This diagram represents variant (A) in Table of Interoperability above.



1. The client application ACI application initiates an RPC request through the SDK: synchronous/conversational or synchronous/non-conversational.
2. Broker stub communicates this request to the broker kernel.
3. a. **Natural**
   The broker kernel communicates this request to Natural nucleus, which behaves like an RPC server for Natural-written applications programs.

   b. **Java**
   Broker communicates this request to RPC server.

   c. **C**
      Broker communicates this request to RPC server.

4. a. **Natural**
      Natural nucleus invokes the RPC server program.

   b. **Java**
      RPC server invokes the server application program.

   c. **C**
      RPC server invokes the server application program.

5. a. **Natural**
      Natural nucleus returns the request to EntireX Broker.

   b. **Java**
      RPC server returns the request to EntireX Broker.

   c. **C**
      RPC server returns the request to EntireX Broker.

6. Broker passes the request to the ACI application.

4 General Architecture of EntireX Broker

Introduction to EntireX Broker Architecture

This section describes the command process flows within the Broker kernel and stubs when two application components communicate with each other using EntireX Broker. The Broker consists of the following components:

- a stub (application binding), which resides within the process space of each application component
- a Broker kernel, which resides in a separate process space, managing all the communication between application components

The details of the transport protocols remain transparent to the application components because they reside within EntireX Broker (stubs and kernel). The EntireX Broker kernel and the location of the transport protocols are the architectural aspects of EntireX Broker that distinguish it from other messaging middleware.

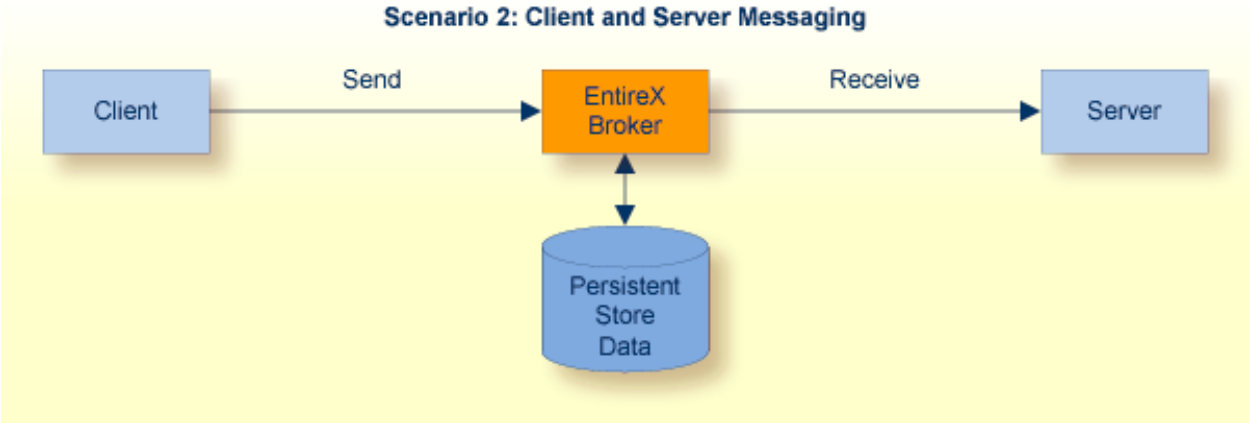
Client Server Communication Model

The EntireX Broker uses the communication model client and server. See *Writing Client and Server Applications* for details.

Example Scenario 1: Client and Server Messaging (Synchronous)

This is a synchronous messaging scenario: send request and wait for a response.

Example Scenario 2: Client and Server Messaging (Asynchronous)



This is an asynchronous messaging scenario: put message in service queue.

Note: Client and server have specific meanings within the context of EntireX.

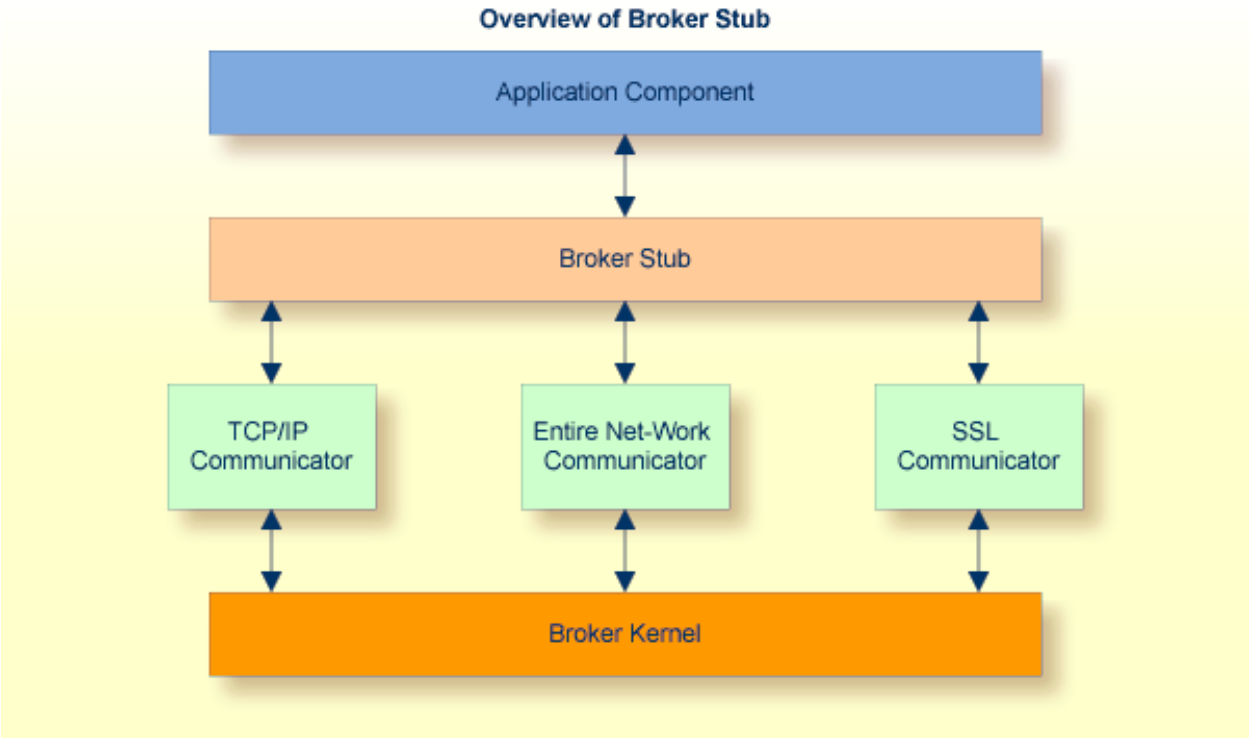
Term	Description
Client	An application component intending to access a service makes its request via EntireX Broker which routes the request to the specific application component offering this service. The request can be a single pair of messages comprising request/reply; or it can be a sequence of multiple, related messages containing one or more requests and one or more replies, known as a conversation. This enables EntireX Broker to be used for applications supporting different programming interfaces. It also allows interoperability between types of application components employing these different interfaces.
Server	An application component offering a service registers it with EntireX Broker. EntireX Broker makes the registered service available to other application components capable of communicating with EntireX Broker. The fact that a server has been registered and is available in this way defines it as a service in terms of class/name/server within the context of EntireX.

Architecture of Broker Stub

The type of communication model described in this section and in the section *Architecture of Broker Kernel* is client and server.

Overview of Broker Stub

The EntireX Broker stub is another name for Software AG's ACI (Advanced Communication Interface). The stub implements an API (application programming interface) that allows programs written in various languages to access EntireX Broker.



See also *Administering Broker Stubs* in the platform-specific Administration documentation.

Description of Command Process Flow within Broker Stub

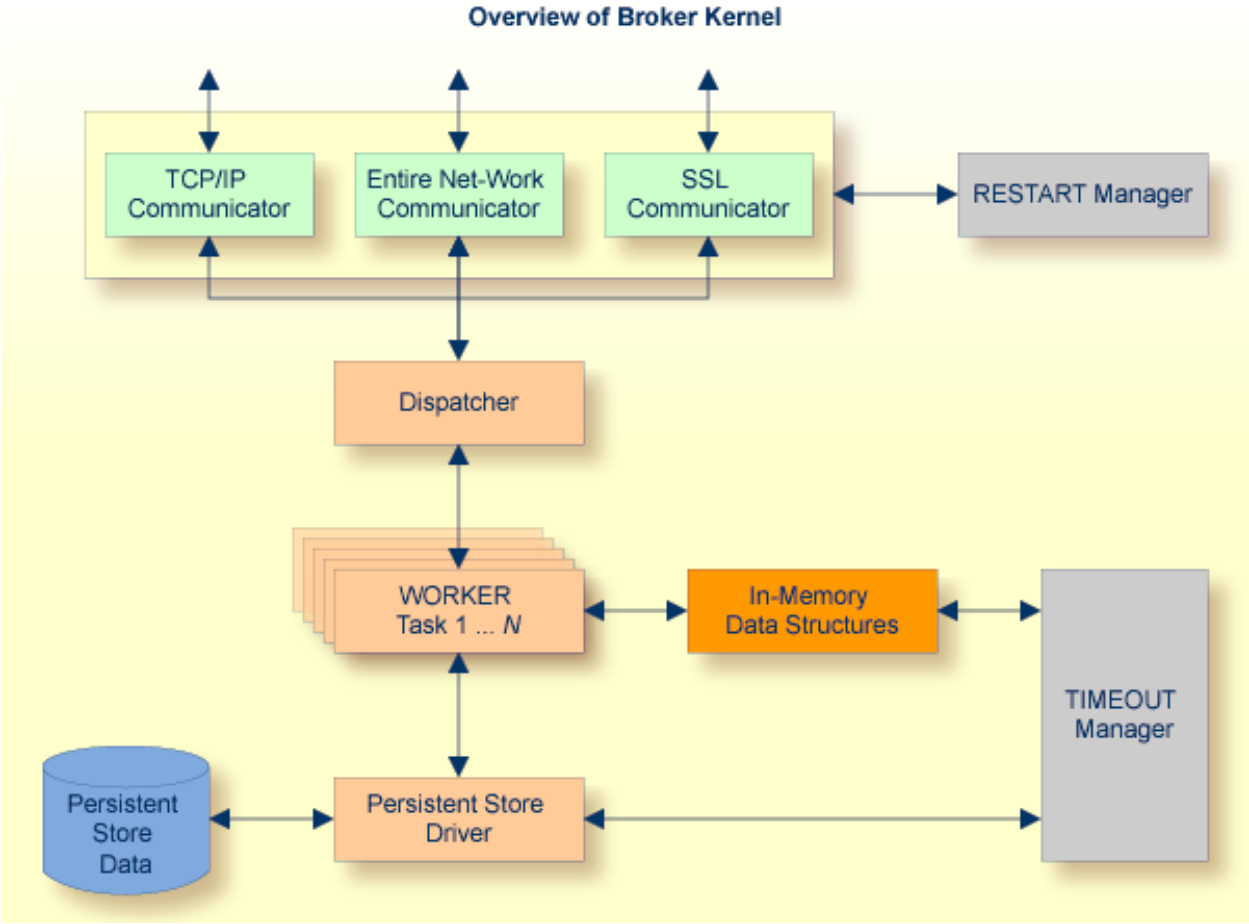
The following table gives a step-by-step description of a typical command process flow from and to a Broker stub. This example describes a `SEND/RECEIVE` command pair.

Step	Description
1	The originating application program calls the stub with a `SEND/WAIT=YES` command. The stub builds the necessary information structures and communicates the message to the Broker kernel. Basic validation is performed in the stub before the command is passed to the Broker kernel.
2	The stub uses one of the following transport mechanisms to transmit the command to the Broker kernel: TCP, SSL or Entire Net-Work. The application does not have to recognize the details of the transport protocol since all transport protocol processing resides entirely within the stub.
3	The application is suspended while the stub waits for a response. Since the application has issued `SEND`, `WAIT=YES` it must wait for the message to travel via the Broker kernel to the partner application which will satisfy the request.
4	After the request has been satisfied and the message returns from the partner application, via the Broker kernel, the stub will pass control back to the originating application.

Architecture of Broker Kernel

The type of communication model described in this section and in the section *Architecture of Broker Stub* is client and server.

Overview of Broker Kernel



Description of Command Process Flow within Broker Kernel

The following table gives a step-by-step description of a typical command process flow within the Broker kernel. This example describes a `SEND/RECEIVE` command pair.

Step	Description
1	The originating application program calls the Broker stub with a `SEND` command. The stub builds the necessary information structures and transmits the message to the Broker kernel using TCP, SSL or Entire Net-Work.
2	The message is received by one of the communications subtasks running within the Broker kernel. The communications subtask passes the message to the dispatcher.
3	The dispatcher schedules the processing of the message within a worker task inside the Broker kernel.
4	Worker task processes the inbound message, performing any necessary data conversion and security operations, and then determines the partner to which the message is to be routed. Any necessary persistence operations are performed under control of the worker task.
5	The outbound message is passed to the relevant communications subtasks within the Broker kernel for transmission to the partner application component.
6	The partner application component which has issued a `RECEIVE` command via the broker stub obtains the message from the originating application program.
7	The partner application component then processes the message and normally makes a reply.

Notes:

1. Application components can exchange successive related message pairs. This action constitutes a conversation.
2. Clean-up processing of timed-out commands is performed asynchronously by the Broker kernel Timeout Manager which acts upon in-memory data structures as well as data within the persistent store.
3. The communications restart manager is able to restart any communications subtasks which may have become temporarily disabled, for example by restarting the machine's TCP/IP driver.

5 Functionality of EntireX Broker

This chapter gives an overview of the major value-added services provided by EntireX Broker. These services relieve the administrator or application builder of the task of providing the desired functionality.

Application Bindings (Stubs)

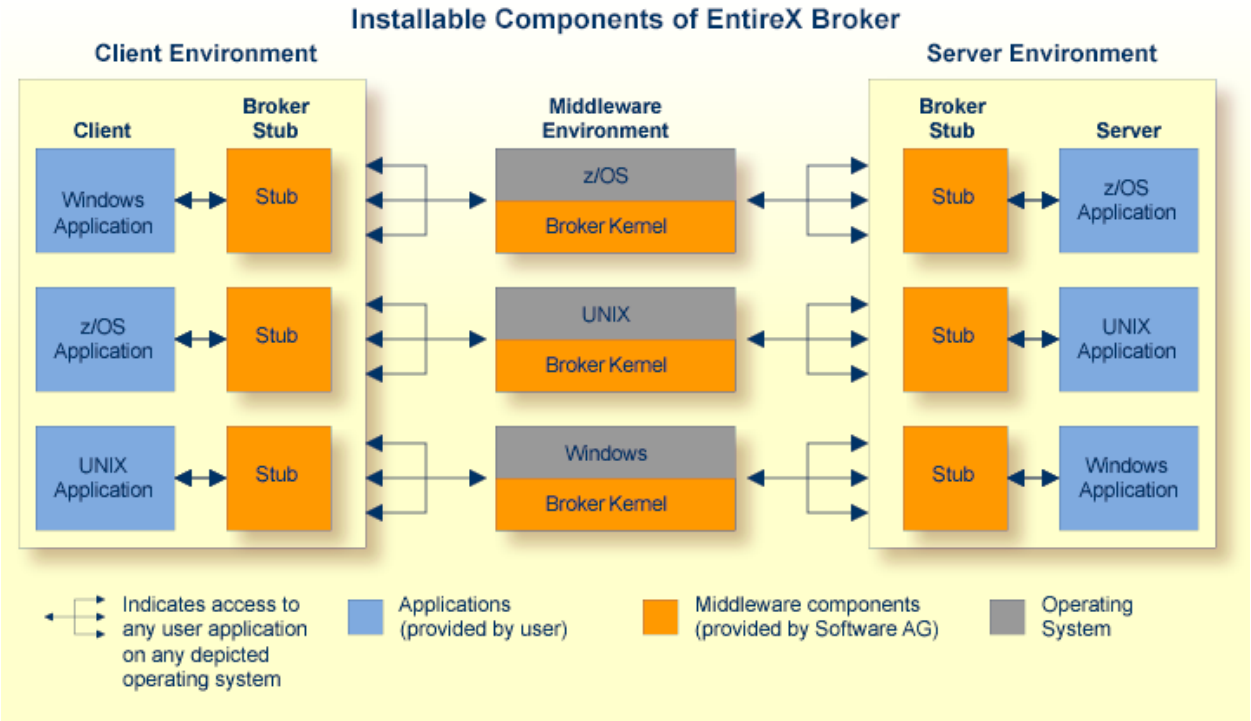
Application bindings allow applications developed in different programming languages and executing on various different platforms to be enabled by using EntireX Broker, see *Architecture of Broker Stub*. Specifically, almost all 3GL, Java and Natural programs are easily enabled using EntireX Broker. These bindings are available on all major mainframe, UNIX and Windows platforms. In addition, the SDK provided by EntireX allows different programming interfaces to be utilized, including COM, RPC and .NET, in addition to EntireX Broker's native programming interface, the Broker ACI.

The application binding - and SDK component, where appropriate - is the glue between the application and the EntireX Broker kernel (see *Architecture of Broker Kernel*, allowing your application to leverage all the functionality of EntireX regardless of

- programming language
- operating system
- hardware platform
- transport mechanism and
- choice of programming interfaces.

This binding capability enables various different application components to be integrated in a loosely coupled manner.

These are the locations where EntireX Broker stubs can be installed:



Attach Services

EntireX Broker provides a choice of mechanisms which enable application components to be started automatically when required.

Example: A client application requires some processing from a server application component. The range of attach services includes starting IMS TM and CICS transactions on the mainframe, and batch programs/processes on mainframe, UNIX and Windows.

Codepage Conversion

Software internationalization is the process of designing products and services so that they can be adapted easily to a variety of different local languages and cultures. Codepage conversion within the EntireX Broker facilitates the internationalization of messages: the incoming and outgoing data is converted to the desired codepage of the platform in use.

Command and Information Services

EntireX Broker includes a set of monitoring and control functions that enable you to monitor system resource utilization and view the current activities of the clients and servers on the system. These services are available through a Web-based interface, in addition to a command-line tool. An interface exists to allow program access to these facilities.

Accounting

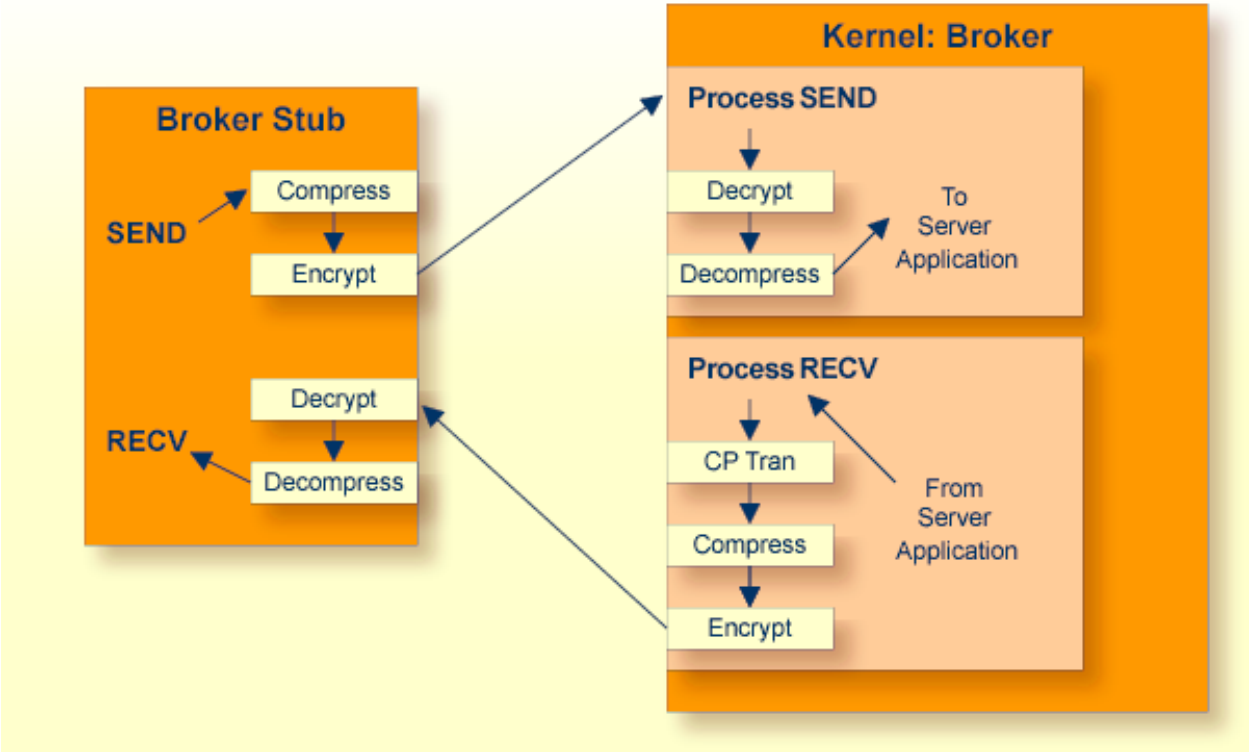
EntireX Broker provides accounting information based upon the flow of message sequences (or conversations). On z/OS, this information is written to standard accounting (SMF) records; on other platforms it is written to a file. The information can be used for:

- application chargeback: apportioning EntireX resource consumption on the conversation and/or the application level;
- performance measurement: analyzing application throughput (bytes, messages, etc.) to determine overall performance;
- trend analysis: using data to determine periods of heavy and/or light resource and/or application usage.

Data Compression

EntireX allows compression of messages passed between application components so as to consume less network bandwidth. This is done independently of transport mechanism by compressing the message in the application binding before it is transmitted to the **EntireX Broker kernel**. The Broker kernel decompresses the message to enable security and data conversion to be applied.

The following graphic illustrates the sequencing of data compression within the stub and Broker kernel:



Persistent Store

The persistent store stores units of work for client and server applications.

Persistent message delivery ensures that messages sent between client and server (or server and client) application components can reach their target even in the event of application or system failures. The user application programs units of work to achieve persistent messaging. EntireX Broker provides persistent message delivery by grouping messages into units of work (UOWs) that are committed in one atomic operation by the sender. See also *Units of Work*.

Persistence is implemented centrally within the **EntireX Broker kernel**. Therefore, the consistency of all the stored messages is guaranteed independently of the different application components and platforms from which the messages are derived.

Persistent Store Types

A persistent store driver is an executable, or a load module, which implements access to the physical persistent store. EntireX Broker allows the choice of three persistent store repositories: Adabas (DBMS), Data In Virtual (DIV) for z/OS, and native file system. The following table gives an overview of the persistent store options:

Persistent Store Type	Description	Operating System	Notes
Adabas	Uses Adabas database.	UNIX, Windows, z/OS, z/VSE	Adabas, Software AG's ADAptable dataBASe, is a high-performance, multithreaded, database management system.
DIV	Uses IBM Data In Virtual facility on z/OS.	z/OS	This persistent store option is implemented as a VSAM linear data set.
CTREE	c-tree© is an embedded local database that can be used as your persistent store.	UNIX and Windows	c-tree© is the fast and reliable embedded database of FairCom Corporation®.

Units of Work

Units of work inform the sender of messages about their past and current status. Specifically, UOWs are used to:

- commit the sending of messages;
- acknowledge the receipt of messages;
- track the progress of sent messages at any point in time.

Units of work are also the vehicle for achieving persistent messaging, although UOWs can be used without persistence.

See also *Using Units of Work*.

Security

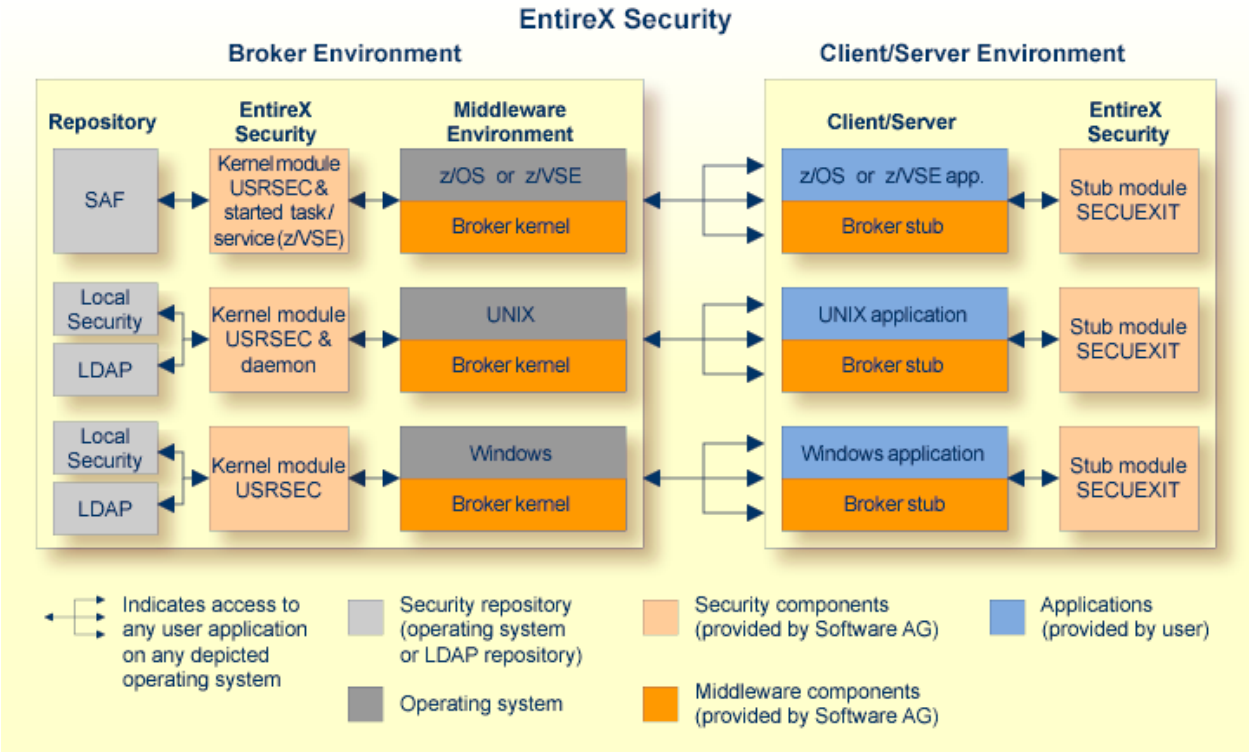
EntireX Security enables distributed application components running with Broker to be executed securely. EntireX Security is located centrally in the kernel of EntireX Broker giving it an overview of all messages sent between application components and therefore providing complete control over the authentication and authorization of each component.

Security checks are performed using a choice of security repositories, including:

- RACF
- CA ACF2
- CA Top Secret
- UNIX and Windows security systems

The security repository chosen depends on the location of the Broker kernel. Because EntireX was designed to operate together with a security system, there is no additional application programming necessary.

This diagram depicts the location of the security components of the kernel and stubs of EntireX Broker:



See also *Security Solutions in EntireX*.

6 Broker Quick Reference

ACI Syntax of Messaging Model

This table provides the ACI syntax used in EntireX Broker's communication model *Client and Server*.

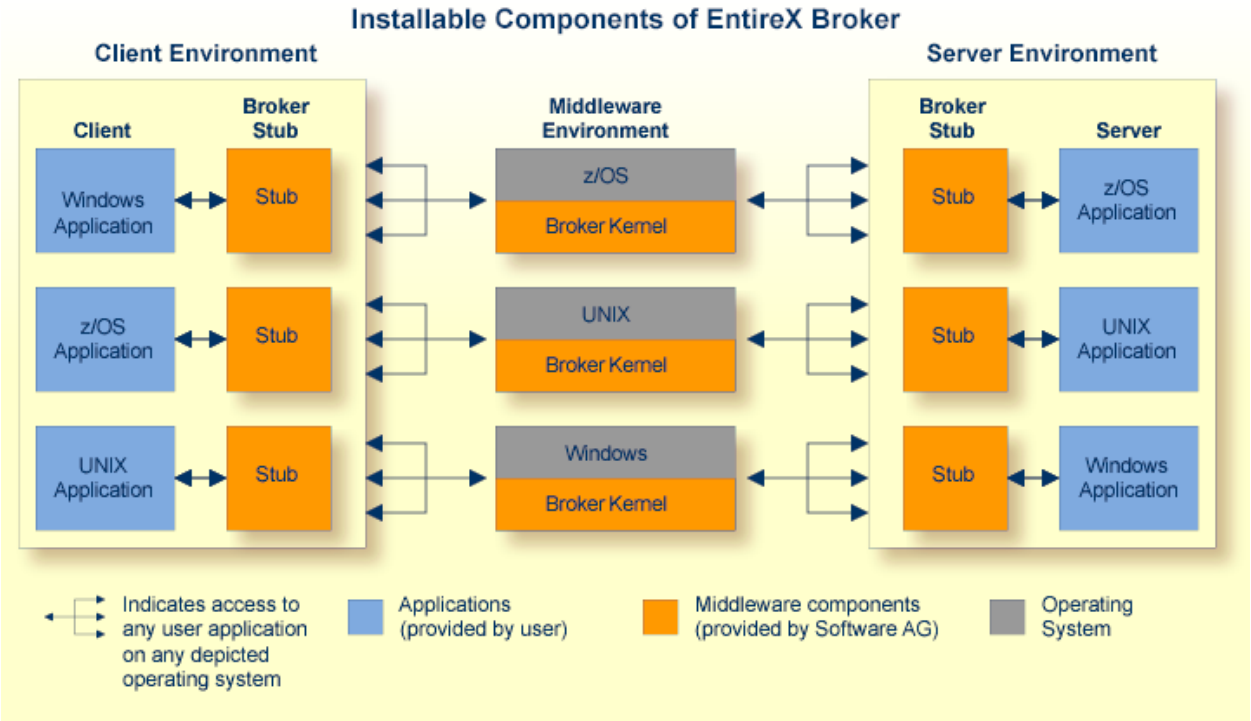
Messaging Term		Client	Server
SYNCHRONICITY	Synchronous	■ SEND (1) ■ WAIT=YES (1)	■ RECEIVE ■ WAIT=YES
	Asynchronous (2)	■ SEND ■ WAIT=NO ■ WAIT=YES	■ RECEIVE ■ WAIT=NO
CONVERSATIONALITY	Conversational (2)	■ SEND ■ CONV-ID=NEW	■ RECEIVE
	Non-conversational (2)	■ SEND ■ CONV-ID=NONE	■ RECEIVE

Notes:

1. The synchronous `SEND, WAIT=YES` command contains an implied `RECEIVE` command.
2. Persistence available. See *Concepts of Persistent Messaging*.

Location of Broker Kernel and Stubs

This graphic shows the locations where the broker kernel and broker stubs can be installed. See *Architecture of Broker Kernel* and *Architecture of Broker Stub*.



Transport: Broker Stubs and APIs

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

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

Notes:

1. NET is available for transport to a broker running under mainframe platforms only; not to a broker running under UNIX or Windows.
2. Under z/OS, use IBM's Application Transparent Transport Layer Security (AT-TLS). Refer to the IBM documentation for more information. See also *SSL/TLS and Certificates with EntireX*.
3. Tracing and transport timeout are not supported in this environment.
4. Stub broker32.dll is supported for reasons of backward compatibility. The functionality is identical to broker.dll.
5. Via EntireX Broker HTTP(S) Agent; see *Settting up and Administering the EntireX Broker HTTP(S) Agent* in the UNIX and Windows Administration documentation.
6. Under z/VSE, use BSI's Automatic Transport Layer Security (ATLS). Refer to the *BSI SSL Installation, Programming and User's Guide*. See also *SSL/TLS and Certificates with EntireX*.

See also:

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

II Broker Attributes

7 Broker Attributes

Note: This section lists all EntireX Broker parameters. Not all parameters are applicable to all supported operating systems.

The Broker attribute file contains a series of parameters (attributes) that control the availability and characteristics of clients and servers, as well as of the Broker itself. You can customize the Broker environment by modifying the attribute settings.

Name and Location of Attribute File

The name and location of the broker attribute file is platform-dependent.

Platform	File Name/Location
z/OS	Member *EXBATTR* in the EntireX Broker source library.
UNIX	File *etbfile* in directory <*InstDir*>/EntireX/config/etb/<*BrokerName*> (default) *
Windows	File <*BrokerName*>.atr in directory <*InstDir*>\EntireX\config\etb\<*BrokerName*> (default) *
BS2000	File *ETB-ATTR* in library *EXX912.JOBS*.
z/VSE	Library member *ETBnnn.ATR*, where *nnn* is a placeholder specifying the broker instance (e.g. *nnn*= the assigned broker ID).

* When starting a broker manually, name and location of the broker attribute file can be overwritten with the environment variable `ETB_ATTR`.

Attribute Syntax

Each entry in the attribute file has the format:

```
ATTRIBUTE-NAME=value
```

The following rules and restrictions apply:

- A line can contain multiple entries separated by commas.
- Attribute names can be entered in mixed upper and lowercase.
- Spaces between attribute names, values and separators are ignored.
- Spaces in the attribute names are not allowed.
- Commas and equal signs are not allowed in value notations.
- Lines starting with an asterisk (*) are treated as comment lines. Within a line, characters following an * or # sign are also treated as comments.
- The `CLASS` keyword must be the first keyword in a service definition.
- Multiple services can be included in a single service definition section. The attribute settings will apply to all services defined in the section.
- Attributes specified after the service definition (`CLASS`, `SERVER`, `SERVICE` *keywords*) overwrite the default characteristics for the service.

- Attribute values can contain variables of the form `${variable name}` or `$variable name`:
  - Due to variations in EBCDIC codepages, braces should only be used on ASCII (UNIX or Windows) platforms or EBCDIC platforms using the IBM-1047 (US) codepage.
  - The variable name can contain only alphanumeric characters and the underscore (_) character.
  - The first non-alphanumeric or underscore character terminates the variable name.
  - Under UNIX and Windows, the string `${variable name}` is replaced with the value of the corresponding environment variable.
  - On z/OS, variable values are read from a file defined by the DD name `ETBVARS`. The syntax of this file is the same as the attribute file.
  - If a variable has no value: if the variable name is enclosed in braces, error 00210594 is given, otherwise `$variable name` will be used as the variable value.
  - If you encounter problems with braces (and this is quite possible in a z/OS environment), we suggest you omit the braces.

Broker-specific Attributes

The broker-specific attribute section begins with the keyword `DEFAULTS=BROKER`. It contains attributes that apply to the broker. At startup time, the attributes are read and duplicate or missing values are treated as errors. When an error occurs, the broker stops execution until the problem is corrected.

Tip: To avoid resource shortages for your applications, be sure to specify sufficiently large values for the broker attributes that define the global resources.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`ABEND-LOOP-DETECTION`	YES | NO	O	z	u	w	v	b
	YES Stop broker if a task terminates abnormally twice, that is, the same abend reason at the same abend location already occurred. This attribute prevents an infinite abend loop. NO Use only if requested by Software AG Support. This setting may make sense if a known error leads to an abnormal termination, but a hotfix solving the problem has not yet been provided. Reset to "YES" when the hotfix has been installed.						
`ABEND-MEMORY-DUMP`	YES | NO	O	z	u	w	v	b
	YES Print all data pools of the broker if a task terminates abnormally. This dump is needed to analyze the abend. NO If the dump has already been sent to Software AG, you can set to "NO" to avoid the extra overhead.						
`ACCOUNTING`	NO | 128-255	O	z				
	NO | YES [SEPARATOR=*char*]	O		u	w	v	b
	Determines whether accounting records are created. NO Do not create accounting records. *nnn* The SMF record number to use when writing the accounting records. YES Create accounting data. *char*= separator character(s). Up to seven separator characters can be specified using the `SEPARATOR` suboption, for example `ACCOUNTING = (YES, SEPARATOR=;)`. If no separator character is specified, the comma character will be used.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	See also *Accounting in EntireX Broker* in the platform-specific Administration documentation.						
ACCOUNTING-VERSION	1 | 2 | 3 | 4 | 5	O	z	u	w	v	b
	Determines whether accounting records are created. 1 Collect accounting information. This value is supported for reasons of compatibility with EntireX Broker 7.2.1 and below. 2 Collect extended accounting information in addition to that available with option 1. 3 Create accounting records in layout of version 3. 4 Create accounting records in layout of version 4. 5 Create accounting records in layout of version 5. This parameter applies when ACCOUNTING is activated.						
APPLICATION-MONITORING or APPMON	YES | NO	O	z	u	w	v	
	Enable application monitoring in EntireX Broker. YES Enable application monitoring. NO Disable application monitoring. See *Application Monitoring*.						
AUTOLOGON	YES | NO	O	z	u	w	v	b
	YES LOGON occurs automatically during the first SEND or REGISTER. NO The application has to issue a LOGON call.						
BLACKLIST-PENALTY-TIME	5m | *n* | *n* S | *n* M | *n* H	R	z	u	w	v	b
	Define the length of time a participant is placed on the PARTICIPANT-BLACKLIST to prevent a denial-of-service attack. *n* Same as *n* S. *n* S Non-activity time in seconds (max. 2147483647). *n* M Non-activity time in minutes (max. 35791394). *n* H Non-activity time in hours (max. 596523). See *Protecting a Broker against Denial-of-Service Attacks* in the platform-specific broker Administration documentation.						
BROKER-ID	A32	R	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Identifies the broker to which the attribute file applies. The broker ID must be unique per machine. **Note:** The numerical section of the `BROKER-ID` is no longer used to determine the DBID in the EntireX Broker kernel with Entire Net-Work transport (NET). To determine the DBID, use attribute `NODE` in the `DEFAULTS=NET` section of the attribute file.						
`CLIENT-NONACT`	15M | *n* | *n*S | *n*M | *n*H	R	z	u	w	v	b
	Define the non-activity time for clients. *n* Same as *n*S. *n*S Non-activity time in seconds (max. 2147483647). *n*M Non-activity time in minutes (max. 35791394). *n*H Non-activity time in hours (max. 596523). A client that does not issue a broker request within the specified time limit is treated as inactive and all resources for the client are freed.						
`CMDLOG`	NO | YES	O	z	u	w	v	b
	NO Command logging will not be available in the broker. YES Command logging features will be available in the broker.						
`CMDLOG-FILE-SIZE`	1024 | *n*	O	z	u	w	v	b
	Defines the maximum size of the file that the command log is written to, in kilobytes. The value must be 1024 or higher. The default value is 1024. When one command log file grows to this size, broker starts writing to the other file. For more details, see *Command Logging in EntireX*.						
`CONTROL-INTERVAL`	60s | *n* | *n*S | *n*M | *n*H |	O	z	u	w	v	b
	Defines the time interval of time-driven broker-to-broker calls. 1. It controls the time between handshake attempts. 2. The standby broker will check the status of the standard broker after the elapsed `CONTROL-INTERVAL` time. *n* Same as *n*S. *n*S Interval in seconds (max. 2147483647). *n*M Interval in minutes (max. 35791394).						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	*n*H Interval in hours (max. 596523). The minimum value is 16 seconds. We strongly recommend the default value (60 seconds), except for very slow machines.						
CONV-DEFAULT	UNLIM | *n*	O	z	u	w	v	b
	Default number of conversations that are allocated for every service. UNLIM The number of conversations is restricted only by the number of conversations globally available. Precludes the use of NUM-CONVERSATION. *n* Number of conversations. This value can be overridden by specifying a CONV-LIMIT for the service. A value of 0 (zero) is invalid.						
DEFERRED	NO | YES	O	z	u	w	v	b
	Disable or enable deferred processing of units of work. NO Units of work cannot be sent to the service until it is available. YES Units of work can be sent to a service that is not up and registered. They will be processed when the service becomes available.						
DYNAMIC-MEMORY-MANAGEMENT	YES | NO	O	z	u	w	v	b
	YES An initial portion of memory is allocated at broker startup based on defined NUM-* attributes or internal default values if no NUM-* attributes have been defined. More memory is allocated without broker restart if there is a need to use more storage. Unused memory is deallocated. The upper limit of memory consumption can be defined by the attribute MAX-MEMORY. See *Dynamic Memory Management* under *Broker Resource Allocation*. NO All memory is allocated at broker startup based on the calculation from the defined NUM-* attributes. Size of memory cannot be changed. This was the known behavior of EntireX 7.3 and earlier. If you run your broker with attribute DYNAMIC-MEMORY-MANAGEMENT=YES, the following attributes are not needed: ■ CONV-DEFAULT ■ NUM-SERVER ■ HEAP-SIZE ■ NUM-SERVICE-EXTENSION ■ LONG-BUFFER-DEFAULT ■ NUM-SERVICE						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	■ SERVER-DEFAULT ■ SHORT-BUFFER-DEFAULT ■ NUM-CLIENT ■ NUM-CMDLOG-FILTER ■ NUM-COMBUF ■ NUM-CONV[ERSATION] ■ NUM-LONG[-BUFFER] ■ NUM-SHORT[-BUFFER] ■ NUM-UOW|MAX-UOWS|MUOW ■ NUM-WQE **Caution:** However, if one of these attributes is defined, it determines the allocation size of that particular broker resource.							
DYNAMIC-WORKER-MANAGEMENT	NO | YES	O	z	u	w		b
	NO All worker tasks are started at broker startup. The number of worker tasks is defined by NUM-WORKER. After this initial step, no further worker tasks can be started. This is default and simulates the behavior of EntireX version 8.0 and earlier. YES As above, the initial portion of worker tasks started at broker startup is determined by NUM-WORKER. However, if there is a need to handle an increased workload, additional worker tasks can be started at runtime without restarting broker. Conversely, if a worker task remains unused, it is stopped. The upper and lower limit of running worker tasks can be defined by the attributes WORKER-MIN and WORKER-MAX. If you run broker with DYNAMIC-WORKER-MANAGEMENT=YES, the following attributes are useful to optimize the overall processing: ■ WORKER-MAX ■ WORKER-MIN ■ WORKER-NONACT ■ WORKER-QUEUE-DEPTH ■ WORKER-START-DELAY The attribute NUM-WORKER defines the initial number of worker tasks started during initialization. See *Dynamic Worker Management* under *Broker Resource Allocation*.							
FORCE	NO | YES	O		u			

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	NO Go down with error if IPC resources still exist. YES Clean up the left-over IPC resources of a previous run. **Note:** 1. If broker is started twice, the second instance will kill the first by removing the IPC resources. 2. For BS2000, z/OS and z/VSE, see separate attribute `FORCE` under `DEFAULTS=NET`.						
`HEAP-SIZE`	1024 | *n*	O	z	u	w	v	b
	Defines the size of the internal heap in KB. Not required if you are using `DYNAMIC-MEMORY-MANAGEMENT`. If you are *not* using dynamic memory management, we strongly recommend specifying - as a minimum - the default value of 1024 KB.						
`ICU-CONVERSION`	YES | NO	O	z	u	w	v	b
	Disable or enable ICU conversion. Default for z/VSE is `NO`; other platforms `YES`. YES ICU is loaded and available for conversion. It is a prerequisite for SAGTCHA and SAGTRPC. NO ICU is not loaded and not available for conversion. SAGTCHA and SAGTRPC cannot be used. If any of the broker service definitions uses the internationalization approach “ICU conversion”, that is, the conversion methods SAGTCHA and SAGTRPC are defined by the service-specific attribute `CONVERSION`, `ICU-CONVERSION` must be set to "YES". The internationalization approaches “Translation”, “Translation User Exit” and “SAGTRPC User Exit” do not require ICU conversion. If all broker service definitions use these internationalization approaches, `ICU-CONVERSION` can be set to "NO". ICU requires additional storage to run properly. If ICU conversion is not needed, setting `ICU-CONVERSION` to "NO" will help to avoid unnecessary storage consumption.						
`ICU-DATA-DIRECTORY`	Folder or directory name in quotes.	O	z	u	w		
	The location where the broker searches for ICU custom converters. See *Building and Installing ICU Custom Converters* in the platform-specific Administration documentation.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
ICU-SET-DATA-DIRECTORY	YES | NO	O	z	u	w		
	Disable or enable ICU custom converter usage. YES The broker tries to locate ICU custom converters with the mechanism defined by the platform, see *Building and Installing ICU Custom Converters* in the platform-specific Administration documentation. NO Use of ICU custom converters is not possible.						
IPV6	YES | NO	O	z	u	w		b
	YES Establish SSL and TCP/IP transport in IPv6 and IPv4 networks according to the TCP/IP stack configuration. NO Establish SSL and TCP/IP transport in IPv4 network only. This attribute applies to EntireX version 9.0 and above.						
LONG-BUFFER-DEFAULT	UNLIM | *n*	O	z	u	w	v	b
	Number of long buffers to be allocated for each service. UNLIM The number of long message buffers is restricted only by the number of buffers globally available. Precludes the use of NUM-LONG-BUFFER. *n* Number of buffers. This value can be overridden by specifying a LONG-BUFFER-LIMIT for the service. A value of 0 (zero) is invalid.						
MAX-MEMORY	0 | *n* | *n*K | *n*M | *n*G | UNLIM	O	z	u	w	v	b
	Defines the upper limit of memory allocated by broker if DYNAMIC-MEMORY-MANAGEMENT=YES has been defined. 0, UNLIM No memory limit. others Defines the maximum limit of allocated memory. If limit is exceeded, error 671 "Requested allocation exceeds MAX-MEMORY" is generated.						
MAX-MESSAGE-LENGTH	2147483647 | *n*	O	z	u	w	v	b
	Maximum message size that the broker kernel can process. This value is transport-dependent. The default value represents the highest positive number that can be stored in a four-byte integer.						
MAX-MESSAGES-IN-UOW	16 | *n*	O	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Maximum number of messages in a UOW.						
MAX-MSG	See MAX-MESSAGE-LENGTH.						
MAX-TRACE-FILES	4 | *n*	O		u	w		
	Defines the number of backup copies of the trace file ETB.LOG. Minimum number is 1; maximum is 999. A new trace file is allocated when the value for TRACE-FILE-SIZE is exceeded. These two attributes prevent a constantly growing ETB.LOG file. See *Trace File Handling* in the UNIX and Windows Administration documentation.						
MAX-UOW-MESSAGE-LENGTH	See MAX-MESSAGE-LENGTH.						
MAX-UOWS	0 | *n*	O	z	u	w	v	b
	The maximum number of UOWs that can be concurrently active broker-wide. The default value is 0 (zero), which means that the broker will process only messages that are not part of a unit of work. If UOW processing is to be done by any service, a MAX-UOWS value must be 1 or larger for the broker. The MAX-UOWS value for the service will default to the value set for the broker. NUM-UOW is an alias of this parameter.						
MESSAGE-CASE	NONE | UPPER | LOWER	O	z	u	w	v	b
	Indicates if certain error message texts returned by the broker to its clients or written by the broker to its log file are to be in mixed case, uppercase, or lowercase. NONE No changes are made to message case. UPPER Messages are changed to uppercase. LOWER Messages are changed to lowercase.						
MUOW	See NUM-UOW.						
NEW-UOW-MESSAGES	YES | NO	O	z	u	w	v	b
	YES New UOW messages are allowed. NO New UOW messages are not allowed. This applies to UOW when using Persistence and should not be used for non-persistent UOWs. A usage example could be the following: The broker persistent store reaches capacity and the broker shuts down. You can set NEW-UOW-MESSAGES to "NO" to prevent new UOW messages from being added after a broker restart. This action allows only consumption (not production) of UOWs to occur after broker restart. After the persistent						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	store capacity has been sufficiently reduced, the EntireX Broker administrator can issue a CIS command, see ALLOW-NEWUOWMSGS. This action allows new UOW messages to be sent to the broker. Reset attribute NEW-UOW-MESSAGES to "YES", which permits new UOW messages to be produced in subsequent broker sessions.						
NUM-BLACKLIST-ENTRIES	256 | *n*	O	z	u	w	v	b
	Number of entries in the participant blacklist. Default value is 256 entries. Together with BLACKLIST-PENALTY-TIME and PARTICIPANT-BLACKLIST, this attribute is used to protect a broker running with SECURITY=YES against denial-of-service attacks. See *Protecting a Broker against Denial-of-Service Attacks* in the platform-specific broker Administration documentation.						
NUM-CLIENT	*n*	R	z	u	w	v	b
	Number of clients that can access the broker concurrently. A value of 0 (zero) is invalid.						
NUM-CMDLOG-FILTER	1 | *n*	O	z	u	w	v	b
	Maximum number of filters that can be specified simultaneously. **Tip:** We recommend you limit this value to the number of services that are being monitored. Minimum value is 1. A value of zero is invalid when the attribute CMDLOG is set to "YES". See ***Command Logging in EntireX*** for more information.						
NUM-COMBUF	1024 | 1 - 999999	R	z	u	w	v	b
	Determines the maximum number of communication buffers available for processing commands arriving in the broker kernel. The size of one communication buffer is usually 16 KB split into 32 slots of 512 bytes, but it ultimately depends on the hardware architecture of your CPU. A value of 0 (zero) is invalid.						
NUM-CONVERSATION or NUM-CONV	*n* | AUTO	R	z	u	w	v	b
	Defines the number of conversations that can be active concurrently. The number specified should be high enough to account for both conversational and non-conversational requests. (Non-conversational requests are treated internally as one-conversation requests.) *n* Number of conversations. AUTO Uses the CONV-DEFAULT and the service-specific CONV-LIMIT values to calculate the number of conversations. The values used in the calculation must not be set to "UNLIM". **Note:**						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	1. A value of 0 (zero) is invalid. If a wildcard service is defined in the service-specific section of the attribute file, the value of AUTO is invalid. 2. See *Wildcard Service Definitions* under *Broker Attributes*.						
NUM-LONG-BUFFER or NUM-LONG	4096 | *n* | AUTO	R	z	u	w	v	b
	Defines the number of long message containers. Long message containers have a fixed length of 4096 bytes and are used to store requests that are larger than 2048 bytes. Storing a request of 8192 bytes, for example, would require two long message containers. *n* Number of buffers. AUTO Uses the LONG-BUFFER-DEFAULT and the service-specific LONG-BUFFER-LIMIT values to calculate the number of long message buffers. The values used in the calculation must not be set to "UNLIM". A value of 0 (zero) is invalid. In *non-conversational* mode, message containers are released as soon as the client receives a reply from the server. If no reply is requested, message containers are released as soon as the server receives the client request. In *conversational* mode, the last message received is always kept until a new one is received. **Note:** 1. If a catch-all service is defined in the service-specific section of the attribute file, the value of AUTO is invalid. 2. See *Wildcard Service Definitions* under *Broker Attributes*.						
NUM-PARTICIPANT-EXTENSION	*n*	O	z	u	w	v	b
	Defines the number of participant extensions to link participants as clients and servers. *n* Number of participant extensions *not specified* If this attribute is not set, the default value is calculated based on NUM-CLIENT and NUM-SERVER. A value of 0 (zero) is invalid.						
NUM-SERVER	*n* | AUTO	R	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Defines the number of servers that can offer services concurrently using the broker. This is *not* the number of services that can be registered to the broker (see NUM-SERVICE). *n* Number of servers. AUTO Uses the SERVER-DEFAULT and the service-specific SERVER-LIMIT values to calculate the number of servers. The values used in the calculation must not be set to "UNLIM". **Note:** 1. Setting this value higher than the number of services allows the starting of server replicas that provide the same service. 2. A value of 0 (zero) is invalid. If a wildcard service is defined in the service-specific section of the attribute file, the value of AUTO is invalid. 3. See *Wildcard Service Definitions* under *Broker Attributes*.							
NUM-SERVICE	*n*	R	z	u	w	v	b
	Defines the number of services that can be registered to the broker. This is *not* the number of servers that can offer the services (see NUM-SERVER). A value of 0 (zero) is invalid.						
NUM-SERVICE-EXTENSION	*n* | AUTO	O	z	u	w	v	b
	Defines the number of service extensions to link servers to services. *n* Number of service extensions. AUTO Uses the value specified or calculated for NUM-SERVER + NUM-CLIENT, plus an extra cushion. *not specified* If this attribute is not set, the default value is NUM-SERVER multiplied by NUM-SERVICE. The minimum value is NUM-SERVER. The maximum value is NUM-SERVER multiplied by NUM-SERVICE. Caution is recommended with this attribute: ■ Set this attribute only if the storage resources allocated for service extensions need to be restricted. ■ Note that the value <*n*> allows only the specified number of server instances of <*n*> to be used. ■ Value AUTO will calculate the number of allowed server instances from NUM-SERVER, which itself might be set to AUTO. In this case, this also							

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	considers the value of SERVER-DEFAULT and even the individual SERVER-LIMIT for each service definition (see note below).						
NUM-SHORT-BUFFER or NUM-SHORT	*n* | AUTO	R	z	u	w	v	b
	Defines the number of short message containers. Short message containers have a fixed length of 256 bytes and are used to store requests of no more than 2048 bytes. To store a request of 1024 bytes, for example, would require four short message containers. *n* Number of buffers. AUTO Uses the SHORT-BUFFER-DEFAULT and the service-specific SHORT-BUFFER-LIMIT values to calculate the number of short message buffers. The values used in the calculation must not be set to "UNLIM". **Note:** 1. In *non-conversational* mode, message containers are released as soon as the client receives a reply from the server. If no reply is requested, message containers are released as soon as the server receives the client request. 2. In *conversational* mode, the last message received is always kept until a new one is received. 3. If a wildcard service is defined in the service-specific section of the attribute file, the value of AUTO is invalid. 4. See *Wildcard Service Definitions* under *Broker Attributes*.						
NUM-UOW	0 | *n*	O	z	u	w	v	b
	The maximum number of UOWs that can be concurrently active broker-wide. The default value is 0 (zero), which means that the broker will process only messages that are not part of a unit of work. If UOW processing is to be done by any service, a NUM-UOW value must be 1 or larger for the broker. (MAX-UOWS is an alias for this attribute.) The NUM-UOW value for the service will default to the value set for the broker.						
NUM-WORKER	1 | *n* (max. 10)	R	z	u	w	v	b
	Number of worker tasks that the broker can use. The number of worker tasks determines the number of functions (SEND, RECEIVE, REGISTER, etc.) that can be processed concurrently. At least one worker task is required; this is the default value.						
NUM-WQE	1 - 32768	R	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Maximum number of requests that can be processed by the broker in parallel, over all transport mechanisms. Each broker command is assigned a worker queue element, regardless of the transport mechanism being used. This element is released when the user has received the results of the command, including the case where the command has timed out.						
PARTICIPANT-BLACKLIST	YES | NO	R	z	u	w	v	b
	Determines whether participants attempting a denial-of-service attack on the broker are to be put on a blacklist. YES Create a participant blacklist. NO Do not create a participant blacklist. See *Protecting a Broker against Denial-of-Service Attacks* in the platform-specific broker Administration documentation.						
PARTNER-CLUSTER-ADDRESS	A32	R	z	u	w	v	b
	This is the address of the load/unload broker in transport-method-style. Transport methods TCP and SSL are supported. See *Transport-method-style Broker ID* for more details. This attribute is required if the attribute RUN-MODE is specified.						
PERCENTAGE-FOR-CONNECTION-SHORTAGE-MESSAGE	90 | 1-100	O	z	u	w	v	b
	Broker will issue a message if the defined percentage value of TCP/IP connections (available file descriptors) is exceeded. Default is 90 percent of the available file descriptors.						
POLL	YES | NO	O	z	u		v	
	In earlier EntireX versions, the maximum number of TCP/IP connections per communicator was limited; see *Maximum TCP/IP Connections per Communicator* under *Broker Resource Allocation* for platform-specific list. With attribute POLL introduced in EntireX version 9.0, this restriction can be lifted under z/OS, UNIX and z/VSE. NO This setting is used to run the compatibility mode in Broker. The poll() system call is not used. The limitations described under *Maximum TCP/IP Connections per Communicator* under *Broker Resource Allocation* apply. YES The poll() system call is used to lift the resource restrictions with select() in multiplexing file descriptor sets.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Note: Setting this attribute to YES increases CPU consumption. POLL=YES is only useful if you need more than the maximum number of TCP/IP connections per communicator; we recommend POLL=NO to reduce CPU consumption.						
PSTORE	NO | HOT | COLD	O	z	u	w	v	b
	Defines the status of the persistent store at broker startup, including the condition of persistent units of work (UOWs). With any value other than "NO", PSTORE-TYPE must be set. NO No persistent store. HOT Persistent UOWs are restored to their prior state during initialization. COLD Persistent UOWs are not restored during initialization, and the persistent store is considered empty. **Note:** For a hot or cold start, the persistent store must be available when your broker is restarted.						
PSTORE-REPORT	NO | YES	O	z	u	w	v	b
	Determines whether PSTORE report is created. NO Do not create the PSTORE report file. YES Create the PSTORE report file. See also *Persistent Store Report*.						
PSTORE-TYPE	DIV (z/OS) | CTREE (UNIX, Windows) | Adabas (all platforms) | FILE (UNIX, Windows)	O	z	u	w	v	b
	Describes the type of persistent store driver required. DIV Data in Virtual. z/OS only, and default on this platform. See *DIV-specific Attributes* below and *Implementing a DIV Persistent Store* under *Managing the Broker Persistent Store*. CTREE c-tree database. UNIX and Windows only. See *c-tree-specific Attributes* and *c-tree Database as Persistent Store* in the UNIX and Windows Administration documentation.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	ADABAS Adabas. All platforms. See also *Adabas-specific Attributes* (below) and *Managing the Broker Persistent Store* in the platform-specific Administration documentation. FILE B-Tree database. UNIX and Windows only. No longer supported.						
PSTORE-VERSION	2 | 3 | 4	O	z	u	w	v	b
	Determines the version of the persistent store. PSTORE=COLD is not needed to upgrade the PSTORE to version 3. Any broker restart with PSTORE-VERSION=3 will upgrade the PSTORE version. PSTORE-VERSION=3 is needed for ICU support. We recommended setting PSTORE-VERSION=3. The DIV PSTORE requires PSTORE-VERSION=4. **Caution:** ■ If you go back to PSTORE-VERSION=2 after upgrading to PSTORE-VERSION=3, the broker will only process data previously created with version 2. No version 3 data will be accessible. ■ If you change the DIV PSTORE from version 3 to 4, perform a COLD restart for the change to take effect, or run PSTORE UNLOAD/LOAD first.						
RUN-MODE	STANDARD | STANDBY | PSTORE-LOAD | PSTORE-UNLOAD	O	z	u	w	v	b
	Determines the initial run mode of the broker. STANDARD Default value. Normal mode. STANDBY Deprecated. Supported for compatibility reasons. PSTORE-LOAD Broker will run as load broker to write Persistent Store data to a new persistent store. See also *Migrating the Persistent Store*. PSTORE-UNLOAD Broker will run as unload broker to read an existing persistent store and pass the data to a broker running in PSTORE-LOAD mode. See also *Migrating the Persistent Store*.						
SECURITY	NO | YES	O	z	u	w	v	b
	Determines whether the EntireX Broker security exits are activated.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	NO The security exits are not activated. YES The security exits are activated. If the security routines cannot be activated, the broker will not start. Broker trace reports the type of security which is active and from where the security module USRSEC is loaded: ■ EntireX Security ■ User-written USRSEC.						
SECURITY-PATH	A255	O	z	u	w		b

	Full path and file name of an executable file (for example, DLL for Windows or shared library for UNIX) containing the user security exit which the kernel will load and call. Example:
	
	`SECURITY-PATH=usersec.dll`
	
	This assumes the DLL is in the default path. Or:
	
	`SECURITY-PATH=c:\brokerexit\yoursecu.dll`
	
	If the path name contains spaces, enclose it in quotation marks. Example:
	
	`SECURITY-PATH="c:\Software AG\broker exit\yoursecu.dll"`
	
	Note: This attribute is used only when implementing a user-written security exit.						

SERVER-DEFAULT	*n* | UNLIM	O	z	u	w	v	b
	Default number of servers that are allowed for every service. *n* Number of servers. UNLIM The number of servers is restricted only by the number of servers globally available. Precludes the use of `NUM-SERVER=AUTO`. This value can be overridden by specifying a `SERVER-LIMIT` for the service. A value of 0 (zero) is invalid.						
SERVICE-UPDATES	YES | NO	O	z	u	w	v	b
	Switch on/off the automatic update mode of the broker.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	YES The broker reads the attribute file whenever a service registers for the first time. This allows the broker to honor modifications in the attribute file *without* a restart. The attribute file is read only when the first server registers for a particular service; it is not reread when a second replica is activated. NO The attribute file is read only once during broker startup. Any changes to the attribute file will be honored only if the broker is restarted.						
SHORT-BUFFER-DEFAULT	UNLIM | *n*	O	z	u	w	v	b
	Number of short buffers to be allocated for each service. UNLIM The number of short message buffers is restricted only by the number of buffers globally available. Precludes the use of NUM-SHORT-BUFFER=AUTO. *n* Number of buffers. This value can be overridden by specifying a SHORT-BUFFER-LIMIT for the service. A value of 0 (zero) is invalid.						
STORAGE-REPORT	NO | YES	O	z	u	w	v	b
	Create a storage report about broker memory usage. NO Do not create the storage report. YES Create the storage report. See *Storage Report*.						
STORE	OFF | BROKER	O	z	u	w	v	b
	Sets the default STORE attribute for all units of work. This attribute can be overridden by the STORE field in the Broker ACI control block. OFF Units of work are not persistent. BROKER Units of work are persistent.						
TRACE-DD	A255	O	z				
	A string containing data set attributes enclosed in quotation marks. These attributes describe the trace output file and must be defined if you are using using a GDG (generation data group) as output data set. See *Flushing Trace Data to a GDG Data Set* under *Tracing EntireX Broker*. The following keywords are supported as part of the TRACE-DD value: ■ DATACLAS						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000

	■ DCB including BLKSIZE, DSORG, LRECL, RECFM
	■ DISP
	■ DSN
	■ MGMTCLAS
	■ SPACE
	■ STORCLAS
	■ UNIT
	
	Refer to your JCL Reference Manual for a complete description of the syntax.
	
	Example:
	
	`TRACE-DD = "DSNAME=EXX.GDG,`
	`DCB=(BLKSIZE=1210,DSORG=PS,LRECL=121,RECFM=FB),`
	`DISP=(NEW,CATLG,CATLG),`
	`SPACE=(CYL,(100,10)),`
	`STORCLAS=SMS"`
	
	Note: If you specify TRACE-DD, you must also specify TRMODE=WRAP and a value for TRBUFNUM for the setting to take effect.							

TRACE-FILE-SIZE	*n* | *n*K | *n*M | *n*G	O		u	w		
	Defines the size of one trace file in kilobytes, megabytes or gigabytes. If this size is exceeded, a new trace file is allocated until the maximum number of trace files specified with MAX-TRACE-FILES is reached. There is no default value. These two parameters help prevent a constantly growing ETB.LOG file. See *Trace File Handling* in the UNIX and Windows Administration documentation.						
TRACE-LEVEL	0 - 4	O	z	u	w	v	b
	The level of tracing to be performed while the broker is running. 0 No tracing. Default value. 1 Traces incoming requests, outgoing replies, resource usage and conversion errors if SAGTRPC is used for CONVERSION with the conversion options SUBSTITUTE-NONCONV or STOP. 2 All of trace level 1, plus all main routines executed. 3 All of trace level 2, plus all routines executed. 4 All of trace level 3, plus Broker ACI control block displays.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use Command Central or the EntireX Broker command-line utility ETBCMD.						
TRANSPORT	TCP-NET | TCP | SSL | NET	O	z			v	b
	TCP | SSL	O		u	w		
	The broker transport may be specified as any combination of one or more of the following methods: TCP TCP/IP is supported. SSL SSL/TLS is supported. NET Entire Net-Work is supported. This value is not supported for a broker under UNIX or Windows. **Examples:** TRANSPORT=NET specifies that only the Entire Net-Work transport method will be supported by the broker. TRANSPORT=TCP-NET specifies that both the TCP/IP and Net-Work transport methods will be supported by the broker. TRANSPORT=TCP-SSL-NET specifies that the TCP/IP, SSL/TLS, and Entire Net-Work transport methods will be supported by the broker. The parameters for each transport method are described in the respective section: TCP | SSL | NET.						
TRAP-ERROR	*nnnn*	O	z	u	w		b
	Where *nnnn* is the four-digit API error number that triggers the trace handler, for example 0007 (Service not registered). Leading zeros are not required. There is no default value. See *Deferred Tracing* in the platform-specific Broker Administration documentation.						
TRBUFNUM	*n*	O	z	u	w		b
	Changes the trace to write trace data to internal trace buffers. *n* is the size of the trace buffer in 64 KB units. There is no default value.						
TRMODE	WRAP	O	z	u	w		b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Changes the trace mode. "WRAP" is the only possible value. This value instructs broker to write the trace buffer (see TRBUFNUM) if an event occurs. This event is triggered by a matching TRAP-ERROR during request processing or when an exception occurs.						
UMSG	See MAX-MESSAGES-IN-UOW.						
UOW-DATA-LIFETIME	1D | *n*S | *n*M | *n*H | *n*D	O	z	u	w	v	b
	Defines the default lifetime for units of work for the service. *n*S Number of seconds the UOW can exist (max. 2147483647). *n*M Number of minutes the UOW can exist (max. 35791394). *n*H Number of hours the UOW can exist (max. 596523). *n*D Number of days the UOW can exist (max. 24855). If the UOW is inactive - that is, is not processed within the time limit - it is deleted and given a status of "TIMEOUT". This attribute can be overridden by the UWTIME field in the Broker ACI control block. See *Timeout Considerations for EntireX Broker*.						
UOW-MSGS	See MAX-MESSAGES-IN-UOW.						
UOW-STATUS-LIFETIME	no value | *n*[S] | *n*M | *n*H | *n*D	O	z	u	w	v	b
	The value to be added to the UOW-DATA-LIFETIME (lifetime of associated UOW). If a value is entered, it must be 1 or greater; a value of 0 will result in an error. If no value is entered, the lifetime of the UOW *status* information will be the same as the lifetime of the UOW itself. *n*S Number of seconds the UOW status exists longer than the UOW itself (max. 2147483647). *n*M Number of minutes (max. 35791394). *n*H Number of hours (max. 596523). *n*D Number of days (max. 24855). The lifetime determines how much additional time the UOW status is retained in the persistent store and is calculated from the time at which the associated UOW enters any of the following statuses: PROCESSED, TIMEOUT, BACKEDOUT, CANCELLED, DISCARDED. The additional lifetime of the UOW status is calculated only when broker is executing. Value in UOW-STATUS-LIFETIME supersedes the value (if specified) in attribute UWSTATP.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Note: If no unit is specified, the default unit is seconds. The unit does not have to be identical to the unit specified for UOW-DATA-LIFETIME.						
UWSTAT-LIFETIME	Alias for UOW-STATUS-LIFETIME.						
UWSTATP	0 | *n*	O	z	u	w	v	b
	Contains a multiplier used to compute the lifetime of a persistent status for the service. The UWSTATP value is multiplied by the UOW-DATA-LIFETIME value (the lifetime of the associated UOW) to determine the length of time the status will be retained in the persistent store. 0 The status is not persistent. 1 - 254 Multiplied by the value of UOW-DATA-LIFETIME to determine how long a persistent status will be retained. **Note:** This attribute has not been supported since EntireX version 7.3. Use UOW-STATUS-LIFETIME instead.						
UWTIME	Alias for UOW-DATA-LIFETIME.						
WAIT-FOR-ACTIVE-PSTORE	NO | YES	O	z	u	w	v	b
	Determines whether broker should wait for the Adabas Persistent Store to become active, or until c-tree PSTORE files become available. NO If broker should start with a PSTORE-TYPE=ADABAS and the database is not active or is not accessible, broker will stop. If broker should start with a PSTORE-TYPE=CTREE and the c-tree files are still in use, broker will stop. YES If broker should start with a PSTORE-TYPE=ADABAS and the database is not active or is not accessible, broker will retry every 10 seconds to initiate communications with the PSTORE. Broker will reject any user requests until it is able to contact the Adabas database. If broker should start with a PSTORE-TYPE=CTREE and the c-tree files are still in use, broker will retry every 10 seconds to rebuild the persistent data. Broker will reject any user requests until it is able to rebuild the persistent data.						
WORKER-MAX	32 | *n* (min. 1, max. 32)	O	z	u	w		b
	Maximum number of worker tasks the broker can use.						
WORKER-MIN	1 | *n* (min. 1, max. 32)	O	z	u	w		b
	Minimum number of worker tasks the broker can use.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
WORKER-NONACT	70S *n* | *n*S | *n*M | nH	O	z	u	w		b
	Non-activity time to elapse before a worker tasks is stopped. *n* Same as *n*S. *n*S Non-activity time in seconds (default 70, max. 2147483647). *n*M Non-activity time in in minutes (max. 35791394). *n*H Non-activity time in hours (max. 596523). **Caution:** A value of 0 (zero) is invalid. If you set this value too low, additional overhead is required for starting and stopping worker tasks. The default and recommended value is 70S.						
WORKER-QUEUE-DEPTH	1 | *n* (min. 1)	O	z	u	w		b
	Number of unassigned user requests in the input queue before another worker task gets started. The default and recommended value is 1. A higher value will result in longer broker response times.						
WORKER-START-DELAY	*internal-value* | *n*	O	z	u	w		b
	n Delay is extended by *n* seconds. Delay after a successful worker task invocation before another worker task can be started to handle current incoming workload. This attribute is used to avoid the risk of recursive invocation of worker tasks, because starting a worker task itself causes workload increase. If no value is specified, an internal value calculated by the broker is used to optimize dynamic worker management. This calculated value is the maximum time required to start a worker task.						

Service-specific Attributes

Each section begins with the keyword DEFAULTS=SERVICE. Services with common attribute values can be grouped together. The attributes defined in the grouping apply to all services specified within it. However, if a different attribute value is defined immediately following the service definition, that new value applies. See also the sections *Wildcard Service Definitions* under *Broker Attributes* and ***Service Update Modes*** below the table.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
APPLICATION-MONITORING or APPMON	YES | NO	O	z	u	w	v	
	YES Enable application monitoring for the specified services. NO Disable application monitoring for the specified services. See *Application Monitoring*.						
APPLICATION-MONITORING-NAME or APPMON-NAME	A100	O	z	u	w	v	
	Specifies the application monitoring name. Used to set the value of the **ApplicationName** KPI. If omitted, the default value from the APPLICATION-MONITORING section is used. If this value is also not specified, the corresponding CLASS/SERVER/SERVICE names are used. See *Application Monitoring*.						
CLASS	A32 (case-sensitive)	R	z	u	w	v	b
	Part of the name that identifies the service together with the SERVER and SERVICE attributes. CLASS must be specified first, followed immediately by SERVER and SERVICE. Classes starting with any of the following are reserved for use by Software AG and should not be used in customer-written applications: BROKER, SAG, ENTIRE, ETB, RPC, ADABAS, NATURAL. Valid characters for class name are letters a-z, A-Z, numbers 0-9, hyphen and underscore. Do not use dollar, percent, period or comma. See also the restriction for SERVICE attribute names.						
CLIENT-RPC-AUTHORIZATION	N | Y	O	z				b
	Determines whether this service is subject to RPC authorization checking.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	N No RPC authorization checking is performed. Y RPC library and program name are appended to the authorization check performed by EntireX Security. Specify "YES" only to RPC-supported services. To allow conformity with Natural Security, the `CLIENT-RPC-AUTHORIZATION` parameter can optionally be defined with a prefix character as follows: `CLIENT-RPC-AUTHORIZATION=(YES,<prefix-character>)`.						
`CONV-LIMIT`	UNLIM | *n*	O	z	u	w	v	b
	Allocates a number of conversations especially for this service. UNLIM The number of conversations is restricted only by the number of conversations globally available. Precludes the use of `NUM-CONVERSATION=AUTO` in the Broker section of the attribute file. *n* Number of conversations. A value of 0 (zero) is invalid. If `NUM-CONVERSATION=AUTO` is specified in the Broker section of the attribute file, `CONV-LIMIT=UNLIM` is not allowed in the service section. A value must be specified or the `CONV-LIMIT` attribute must be suppressed entirely for the service so that the default (`CONV-DEFAULT`) becomes active.						
`CONV-NONACT`	5M | *n* | *n*S | *n*M | *n*H	R	z	u	w	v	b
	Non-activity time for connections. *n* Same as *n*S. *n*S Non-activity time in seconds (max. 2147483647). *n*M Non-activity time in minutes (max. 35791394). *n*H Non-activity time in hours (max. 596523). A value of 0 (zero) is invalid. If a connection is not used for the specified time, that is, a server or a client does not issue a broker request that references the connection in any way, the connection is treated as inactive and the allocated resources are freed.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
CONVERSION	Format: A255 (SAGTCHA [, TRACE =*n*] [, *OPTION* =*s*] | SAGTRPC [, TRACE =*n*] [, *OPTION* =*s*] | *name* [, TRACE =*n*] | NO)	O	z	u	w	v	b

Defines conversion for internationalization. See *Internationalization with EntireX* and *What is the Best Internationalization Approach to use?* for help on making decisions about the internationalization approach.

SAGTCHA [(1)]	Conversion using ICU Conversion for *ACI-based Programming*.
SAGTRPC [(2)]	Conversion using ICU Conversion for *RPC-based Components* and *Reliable RPC*. We recommend always using SAGTRPC for RPC data streams. *Conversion with Multibyte, Double-byte and other Complex Codepages* will always be correct, and *Conversion with Single-byte Codepages* is also efficient because SAGTRPC detects single-byte codepages automatically. See *Conversion Details*.
<*name*> [(3)]	Name of the SAGTRPC user exit for RPC-based components. See also *Configuring SAGTRPC User Exits* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation and *Writing SAGTRPC User Exits* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation.
NO	If conversion is not to be used, either omit the CONVERSION attribute or specify CONVERSION=NO, for example for binary payload.

Only one internationalization approach can be active at one time for a service. The CONVERSION attribute for internationalization overrides the TRANSLATION attribute when defined for a service. That is, when TRANSLATION and CONVERSION are both defined, TRANSLATION will be ignored.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Note: 1. See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. 2. *Conversion with Multibyte, Double-byte and other Complex Codepages* is not supported on BS2000. For *Conversion with Single-byte Codepages*, use SAGTCHA. 3. SAGTRPC user exit is not supported on z/VSE and BS2000. **TRACE** If tracing is switched on, the trace output is written to the broker log file: 0 No tracing 1 Trace level STANDARD — This level is an "on-error" trace. It provides information on conversion errors only. For RPC calls this includes the IDL library, IDL program and the data. Please note that if *OPTION Values for Conversion* are set, errors are ignored. 2 Trace level ADVANCED — Tracing of incoming, outgoing parameters and the payload. 3 Trace level SUPPORT — This trace level is for support diagnostics and should only be switched on when requested by Software AG support. **OPTION** See table of possible values under *OPTION Values for Conversion*.							
DEFERRED	NO | YES	O	z	u	w	v	b
	NO Units of work cannot be sent to the service until it is available. YES Units of work can be sent to a service that is not up and registered. The units of work will be processed when the service becomes available.						
LOAD-BALANCING	YES | NO	O	z	u	w	v	b
	YES When servers that offer a particular service are started, new conversations will be assigned to these servers in a						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	round-robin fashion. The first waiting server will get the first new conversation, the second waiting server will get the second new conversation, and so on. NO A new conversation is always assigned to the first server in the queue.						
LONG-BUFFER-LIMIT	UNLIM | *n*	O	z	u	w	v	b
	Allocates a number of long message buffers for the service. UNLIM The number of long message buffers is restricted only by the number of buffers globally available. Precludes the use of NUM-LONG-BUFFER=AUTO in the Broker section of the attribute file. *n* Number of long message buffers. A value of 0 (zero) is invalid. If NUM-LONG-BUFFER=AUTO is specified in the Broker section of the attribute file, LONG-BUFFER-LIMIT=UNLIM is not allowed in the service section. A value must be specified or the LONG-BUFFER-LIMIT attribute must be suppressed entirely for the service so that the default (LONG-BUFFER-DEFAULT) becomes active.						
MAX-MESSAGES-IN-UOW	16 | *n*	O	z	u	w	v	b
	Maximum number of messages in a UOW.						
MAX-MESSAGE-LENGTH	2147483647 | *n*	O	z	u	w		b
	Maximum message size that can be sent to a service. This is transport-dependent. The default value represents the highest positive number that can be stored in a four-byte integer.						
MAX-MSG	See MAX-MESSAGE-LENGTH.						
MAX-UOW-MESSAGE-LENGTH	See MAX-MESSAGE-LENGTH.						
MAX-UOWS	0 | *n*	O	z	u	w	v	b
	0 The service does not accept units of work, i.e. it processes only messages that are not part of a UOW. Using zero prevents the sending of UOWs to services that are not intended to process them. *n* Maximum number of UOWs that can be active concurrently for the service. If you do not provide a MAX-UOWS value for the service, it defaults to the MAX-UOWS setting for the broker. If you						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	provide a value that exceeds that of the broker, the service MAX-UOWS is set to the broker's MAX-UOWS value and a warning message is issued. Specify MAX-UOWS=0 for Natural RPC Servers. This restriction will be removed with a later release.						
MUOW	See MAX-UOWS.						
NOTIFY-EOC	NO | YES	O	z	u	w	v	b
	Specifies whether timed-out conversations are to be stored or discarded. NO Discard the EOC notifications if the server is not ready to receive. YES Store the EOC notifications if the server is not ready to receive and then notify the server if possible. If a server is not ready to receive an EOC notification, it can be stored or discarded. If it is stored, the server is notified, if possible, when it is ready to receive. **Caution:** The behavior activated by this parameter can be relied upon only during a single lifetime of the broker kernel. Specifically, conversations containing units of work, whose lifetime can span multiple broker kernel sessions, cannot be assumed to show this behavior, even with NOTIFY-EOC=YES.						
NUM-UOW	Alias for MAX-UOWS.						
SERVER	A32 (case-sensitive)	R	z	u	w	v	b
	Part of the name that identifies the service together with the CLASS and SERVICE attributes. CLASS must be specified first, followed immediately by SERVER and SERVICE. Valid characters for server name are letters a-z, A-Z, numbers 0-9, hyphen and underscore. Do not use dollar, percent, period or comma.						
SERVER-DEFAULT	*n* | UNLIM	O	z	u	w	v	b
	Default number of servers that are allowed for every service. *n* Number of servers.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	UNLIM The number of servers is restricted only by the number of servers globally available. Precludes the use of NUM-SERVER=AUTO. A value of 0 (zero) is invalid. This value can be overridden by specifying a SERVER-LIMIT for the service.						
SERVER-LIMIT	*n* | UNLIM	O	z	u	w	v	b
	Allows a number of servers especially for this service. *n* Number of servers. UNLIM The number of servers is restricted only by the number of servers globally available. Precludes the use of NUM-SERVER=AUTO in the Broker section of the attribute file. A value of 0 (zero) is invalid. If NUM-SERVER=AUTO is specified in the Broker section of the attribute file, SERVER-LIMIT=UNLIM is not allowed in the service section. A value must be specified or the SERVER-LIMIT attribute must be suppressed entirely for the service so that the default (SERVER-DEFAULT) becomes active. **Note:** UNIX and Windows: This limit also includes any attach server you are using. Make sure you increase the number by one for each attach server you use.						
SERVER-NONACT	5M | *n* | *n*S | *n*M | *n*H	R	z	u	w	v	b
	Non-activity time for servers. A server that does not issue a broker request within the specified time limit is treated as inactive and all resources for the server are freed. *n* Same as *n*S. *n*S Non-activity time in seconds (max. 2147483647). *n*M Non-activity time in minutes (max. 35791394). *n*H Non-activity time in hours (max. 596523). If a server registers multiple services, the highest value of all the services registered is taken as non-activity time for the server.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
SERVICE	A32 (case-sensitive)	R	z	u	w	v	b
	Part of the name that identifies the service together with the CLASS and SERVER attributes. CLASS must be specified first, followed immediately by SERVER and SERVICE. The SERVICE attribute names "EXTRACTOR" and "DEPLOYMENT" are reserved for Software AG internal use and should not be used in customer-written applications. Valid characters for service name are letters a-z, A-Z, numbers 0-9, hyphen and underscore. Do not use dollar, percent, period or comma. See also the restriction for CLASS attribute names.						
SHORT-BUFFER-LIMIT	UNLIM | *n*	O	z	u	w	v	b
	Allocates a number of short message buffers for the service. UNLIM The number of short message buffers is restricted only by the number of buffers globally available. Precludes the use of NUM-SHORT-BUFFER=AUTO in the Broker section of the attribute file. *n* Number of short message buffers. If NUM-SHORT-BUFFER=AUTO is specified in the Broker section of the attribute file, SHORT-BUFFER-LIMIT=UNLIM is not allowed in the service section. A value must be specified or the SHORT-BUFFER-LIMIT attribute must be suppressed entirely for the service so that the default (SHORT-BUFFER-DEFAULT) becomes active.						
STORE	OFF | BROKER	O	z	u	w	v	b
	Sets the default STORE attribute for all units of work sent to the service. OFF Units of work are not persistent. BROKER Units of work are persistent. This attribute can be overridden by the STORE field in the Broker ACI control block.						
TRANSLATION	Format: A255 SAGTCHA | NO | <*name*>	O	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Activates *translation* or *translation user exit* for internationalization (see *Translation User Exit*). For help on deciding the right internationalization approach for your environment, see *What is the Best Internationalization Approach to use?* SAGTCHA Conversion routine SAGTCHA for *ACI-based Programming*, *RPC-based Components* and *Reliable RPC*. NO If translation is not to be used - e.g., for binary payload (broker messages) - either omit the `TRANSLATION` attribute or specify `TRANSLATION=NO`. *<name>* Name of Translation User Exit. See also *Configuring Translation User Exits* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation or *Writing Translation User Exits* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. The `CONVERSION` attribute for internationalization overrides the `TRANSLATION` attribute when defined for a service; that is, when `TRANSLATION` and `CONVERSION` are both defined, `TRANSLATION` will be ignored.						
`UMSG`	Alias for `MAX-MESSAGES-IN-UOW`.						
`UOW-DATA-LIFETIME`	1D | *n* S | *n* M | *n* H | nD	O	z	u	w	v	b
	Defines the default lifetime for units of work for the service. *n*S Number of seconds the UOW can exist (max. 2147483647). *n*M Number of minutes the UOW can exist (max. 35791394). *n*H Number of hours the UOW can exist (max. 596523). *n*D Number of days the UOW can exist (max. 24855). If the unit of work (UOW) is inactive, that is, not processed within the time limit, it is deleted and given a status of `TIMEOUT`. This attribute can be overridden by the `UWTIME` field in the Broker ACI control block.						
`UOW-MSGS`	Alias for `MAX-MESSAGES-IN-UOW`.						
`UOW-STATUS-LIFETIME`	no value | *n*[S] | *n*M | *n*H | *n*D	O	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	The value to be added to the UOW-DATA-LIFETIME (lifetime of associated UOW). If a value is entered, it must be 1 or greater; a value of 0 will result in an error. If no value is entered, the lifetime of the UOW *status* information will be the same as the lifetime of the UOW itself. *n*S Number of seconds the UOW status exists longer than the UOW itself (max. 2147483647). *n*M Number of minutes (max. 35791394). *n*H Number of hours (max. 596523). *n*D Number of days (max. 24855). The lifetime determines how much additional time the UOW status is retained in the persistent store and is calculated from the time at which the associated UOW enters any of the following statuses: PROCESSED, TIMEOUT, BACKEDOUT, CANCELLED, DISCARDED. The additional lifetime of the UOW status is calculated only when broker is executing. Value in UOW-STATUS-LIFETIME supersedes the value (if specified) in attribute UWSTATP. **Note:** If no unit is specified, the default unit is seconds. The unit does not have to be identical to the unit specified for UOW-DATA-LIFETIME.						
UWSTATP	0 | *n*	O	z	u	w	v	b
	Contains a multiplier used to compute the lifetime of a persistent status for the service. The UWSTATP value is multiplied by the UOW-STATUS-LIFETIME value (the lifetime of the associated UOW) to determine the length of time the status will be retained in the persistent store. 0 The status is not persistent. 1 - 254 Multiplied by the value of UOW-DATA-LIFETIME to determine how long a persistent status will be retained. **Note:** This attribute has not been supported since EntireX version 7.3. Use UOW-STATUS-LIFETIME instead.						
UWSTAT-LIFETIME	Alias for UOW-STATUS-LIFETIME.						
UWTIME	Alias for UOW-DATA-LIFETIME.						

Wildcard Service Definitions

The special names of CLASS = *, SERVER = * and SERVICE = * are allowed in the service-specific and authorization rule-specific sections of the broker attribute file. These are known as "wildcard" service definitions. If this name is present in the attribute file, any service that registers with the broker and does not have its own entry in the attribute file will inherit the attributes that apply to the first wildcard service definition found.

For example, a server that registers with CLASS=ACLASS, SERVER=ASERVER and SERVICE=ASERVICE can inherit attributes from any of the following entries in the attribute file (this list is not necessarily complete):

```
CLASS = *, SERVER = ASERVER, SERVICE = ASERVICE
CLASS = ACLASS, SERVER = *, SERVICE = *
CLASS = *, SERVER = *, SERVICE = *
```

Of course, if there is a set of attributes that are specifically defined for CLASS=ACLASS, SERVER=ASERVER, SERVICE=ASERVICE, then all of the wildcard service definitions will be ignored in favor of the exact matching definition.

Service Update Modes

EntireX has two modes for handling service-specific attributes. See broker-specific attribute SERVICE-UPDATES.

- In **service update mode** (SERVICE-UPDATES=YES), the service configuration sections of the attribute file are read whenever the first replica of a particular service registers.
- In **non-update mode** (SERVICE-UPDATES=NO), the attribute file is not reread. All attributes are read during startup and the broker does not honor any changes in the attribute file. This mode is useful if
  - there is a high frequency of REGISTER operations, or
  - the attribute file is rather large and results in a high I/O rate for the broker.

  The disadvantage to using non-update mode is that if specific attributes are modified, the broker must be restarted to effect the changes. Generally, this mode should be used only if the I/O rate of the broker is considerably high, and if the environment seldom changes.

OPTION Values for Conversion

The different option values allow you to either handle character conversion deficiencies as errors, or to ignore them:

1. Do not ignore any character conversion errors and force an error always (value `STOP`). This is the default behavior.
2. Ignore if characters can not be converted into the receiver's codepage, but force an error if sender characters do not match the sender's codepage (value `SUBSTITUTE-NONCONV`).
3. Ignore any character conversion errors (values `SUBSTITUTE` and `BLANKOUT`).

Situations 1 and 2 above are reported to the broker log file if the `TRACE` option for `CONVERSION` is set to level 1.

Value	Description	Options Supported for		Report Situation in Broker Log File if `TRACE` Option for `CONVERSION` is set to 1	
		SAGTCHA	SAGTRPC	Bad Input Characters (Sender's Codepage)	Non-convertible Characters (Receiver's Codepage)
`SUBSTITUTE`	Substitutes both non-convertible characters (receiver's codepage) and bad input characters (sender's codepage) with a codepage-dependent default replacement character.	yes	yes	No message.	No message
`SUBSTITUTE-NONCONV`	If a corresponding code point is not available in the receiver's codepage, the character cannot be converted and is substituted with a codepage-dependent default replacement character. Bad input characters in sender's codepage are not substituted and result in an error.	yes	yes	Write detailed conversion error message.	No message.
`BLANKOUT`	Substitutes non-convertible characters with a codepage-dependent default replacement; blanks out the complete RPC IDL field containing one or more bad input characters.	no	yes	No message.	No message.

Value	Description	Options Supported for		Report Situation in Broker Log File if TRACE Option for CONVERSION is set to 1	
		SAGTCHA	SAGTRPC	Bad Input Characters (Sender's Codepage)	Non-convertible Characters (Receiver's Codepage)
STOP	Signals an error on detecting a non-convertible or bad input character. This is the default behavior if no option is specified.	yes	yes	Write detailed conversion error message.	Write detailed conversion error message.

Codepage-specific Attributes

The codepage-specific attribute section begins with the keyword `DEFAULTS=CODEPAGE` as shown in the sample attribute file. You can use the attributes in this section to customize the broker's locale string defaults and customize the mapping of locale strings to codepages for the internationalization approaches ICU conversion and SAGTRPC user exit. These attributes do not apply to other approaches. See *Internationalization with EntireX* for more information.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`DEFAULT_ASCII`	Any ICU converter name or alias. See also ***Additional Notes*** below.	O	z	u	w	v	b

	Customize the broker's locale string defaults by assigning the default codepage for EntireX components (client or server). See *Broker's Locale String Defaults*. This value is used instead of the broker's locale string defaults if
	■ the calling component does not send a locale string itself, and
	■ the calling component is running on an ASCII platform (UNIX, Windows, etc.), and
	■ one of the internationalization approaches ICU conversion or SAGTRPC user exit is used. See *ICU Conversion* and *SAGTRPC User Exit*.
	Example:
	`DEFAULTS=CODEPAGE`
	`  * Broker Locale String Defaults`
	`  DEFAULT_ASCII=windows-950`
	For more examples, see *Configuring Broker's Locale String Defaults* and also ***Additional Notes*** below.						

`DEFAULT_EBCDIC_IBM`	Any ICU converter name or alias	O	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000

	Customize the broker's locale string defaults by assigning the default codepage for EntireX components (client or server). See *Broker's Locale String Defaults*. This value is used instead of the broker's locale string defaults if
	■ the calling component does not send a locale string itself and
	■ the calling component is running on an IBM mainframe platform (z/OS, z/VSE etc.) and
	■ one of the internationalization approaches ICU conversion or SAGTRPC user exit is used.
	Example:
	`DEFAULT=CODEPAGE`
	`  DEFAULT_EBCDIC_IBM=ibm-937`
	For more examples, see *Configuring Broker's Locale String Defaults* and also ***Additional Notes*** below.						

`DEFAULT_EBCDIC_SNI`	Any ICU converter name or alias	O	z	u	w	v	b

	Customize the broker's locale string defaults by assigning the default codepage for EntireX components (client or server). See *Broker's Locale String Defaults*. This value is used instead of the locale string defaults if
	■ the calling component does not send a locale string itself, and
	■ the calling component is running on a Fujitsu EBCDIC mainframe platform (BS2000), and
	■ one of the internationalization approaches ICU conversion or SAGTRPC user exit is used.
	Example:
	`DEFAULT=CODEPAGE`
	`  DEFAULT_EBCDIC_SNI= bs2000-edf03drv`
	For more examples, see *Configuring Broker's Locale String Defaults* and also ***Additional Notes*** below.						

`locale-string`	Any ICU converter name or alias. See also	O	z	u	w	v	

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Additional Notes below.						

	Customize the mapping of locale strings to codepages and bypass the broker's locale string processing mechanism. See *Broker's Locale String Processing*. This is useful:
	■ if the broker's locale string processing fails - i.e. leads to no codepage or to the wrong codepage - you can explicitly assign the codepage which meets your requirements.
	■ if you want to install user-written ICU converters (codepages) into the broker, see *Building and Installing ICU Custom Converters* in the platform-specific Administration documentation.
	The attribute (locale string) is the locale string sent by your EntireX component (client or server) and the value is the codepage that you want to use in place of that locale string. In the first line of the example below, the client or server application sends ASCII as a locale string; the broker maps this to the codepage ISO 8859_1. In the same way EUC_JP_LINUX is mapped to ibm-33722_P12A-1999. All other locale strings are mapped by the broker's mapping mechanism, see *Broker's Built-in Locale String Mapping*. Example:
	`DEFAULTS=CODEPAGE`
	`  * Broker Locale String Codepage Assignments`
	`  ASCII=ISO8859`
	`  EUC_JP_LINUX=ibm-33722_P12A-1999`
	`  * Customer-written ICU converters`
	`  CP1140=myebcdic`
	`  CP0819=myascii`
	For more examples, see *Bypassing Broker's Built-in Locale String Mapping* and also *Additional Notes* below.							

Additional Notes

- Locale string matching is case insensitive when bypassing the broker's built-in mechanism, that is, when the broker examines the codepages section in the attribute file.
- If ICU is used for the internationalization approach and if the style in not known by ICU, e.g. ECS*nnnn*, <ll>_<cc> etc., the name will be mapped to a suitable ICU alias. For more details on the mapping mechanism, see *Broker's Built-in Locale String Mapping*. For more details on ICU and ICU converter name standards, see *ICU Resources*.

- If SAGTRPC user exit is used for the internationalization approach, we recommend assigning the codepage in the form CP<*nnnnn*>. To determine the number given to SAGTRPC user exit, see *Broker's Built-in Locale String Mapping*.
- See `CONVERSION` on this page for the internationalization approach in use.

Adabas SVC/Entire Net-Work-specific Attributes

The Adabas SVC/Entire Net-Work-specific attribute section begins with the keyword `DEFAULTS=NET` as shown in the sample attribute file. The attributes in this section are needed to execute the Adabas SVC/Entire Net-Work communicator of the EntireX Broker kernel.

Note: This section applies to mainframe platforms only. It does not apply to UNIX and Windows.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
ADASVC	*nnn*	R	z			v	
	Sets the Adabas SVC number for EntireX Broker access. The Adabas SVC is used to perform various internal functions, including communication between the caller program and EntireX Broker. Not supported on BS2000.						
EXTENDED-ACB-SUPPORT	NO | YES	O	z			v	b
	Determines whether extended features of Adabas version 8 (or above) are supported. NO No features of Adabas version 8 or above will be used. YES Informs broker kernel to provide Adabas/WAL version 8 transport capability. This parameter is required for sending/receiving more than 32 KB data over Adabas [NET] transport. This value should be set only if you have installed Adabas/WAL version 8, Adabas SVC, and included Adabas/WAL version 8 load libraries into the steplib of broker kernel; otherwise, unpredictable results can occur.						
FORCE	NO | YES	O	z			v	b
	Determines whether DBID table entries can be overwritten. NO Overwrite of DBID table entries not permitted. YES Overwrite of DBID table entries permitted. This is required when the DBID table entry is not deleted after abnormal termination. **Caution:** Overwriting an existing entry prevents any further communication with the overwritten node. Use `FORCE=YES` only if you are absolutely sure that no target node with that DBID is active.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
IDTNAME	FORMAT: A8 *idtname* | ADABAS5B	O					b
	If an ID table name is specified with the appropriate ADARUN parameter for Entire Net-Work, Adabas or Natural, the same name must be specified here. The ID table is used to perform various internal functions, including communication between the caller program and the EntireX Broker. Only supported under BS2000.						
IUBL	8000 | *n*	O	z			v	b
	This parameter sets the maximum length (in bytes) of the buffer that can be passed from the caller to EntireX Broker. The maximum size of IUBL is the same as the maximum value of the Adabas parameter LU (see the *Adabas Operations Manual*). IUBL must be large enough to hold the maximum send-length plus receive-length required for any caller program plus any administrative overhead for Adabas and Entire Net-Work control structures.						
LOCAL	NO | YES	O	z			v	b
	For remote nodes accessed via Entire Net-Work, the attribute LOCAL specifies whether the target ID defined with the NODE attribute can be accessed only locally, or also remotely. NO DBID is *global* and can be accessed from remote nodes via Entire Net-Work. YES DBID is *local* and cannot be accessed from remote nodes via Entire Net-Work.						
MAX-MESSAGE-LENGTH	2147483647 | *n*	O	z	u	w	v	b
	Maximum message size that the broker kernel can process using transport method NET. The default value represents the highest positive number that can be stored in a four-byte integer.						
NABS	10 | *n*	O	z			v	b
	The number of attached buffers to be used (max. 524287). An attached buffer is an internal buffer used for interprocess communication. An attached buffer pool equal to the NABS value multiplied by 4096 will be allocated. This buffer pool must be large enough to hold all data (IUBL) of all parallel calls to EntireX Broker. The following formula can be used to calculate the value for NABS: NABS = NCQE *IUBL / 4096.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
NCQE	10 | *n*	O	z			v	b
	NCQE defines the number of command queue elements which are available for processing commands arriving at the broker kernel over Adabas SVC / Net-Work transport mechanism. Sufficient NCQE should be allocated to allow this transport mechanism to process multiple broker commands concurrently. Each command queue element requires 192 bytes, and the element is released when either the user (client or server) has received the results of the command, or if the command is timed out. The number of command queue elements required to handle broker calls depends on the number of parallel active broker calls that are using the transport mechanism Adabas SVC / Entire Net-Work. For example, all broker commands issued by client or server components using this transport mechanism:						
NODE	1-65534	R	z			v	b
	Defines the unique DBID for EntireX Broker. Used for internode Adabas/Entire Net-Work communication. There is no default; the value of NODE must be a value greater than or equal to 1 or less than or equal to 65534. If you set the parameter LOCAL=YES, you can use the same node number for different installations of EntireX Broker in an Entire Net-Work environment.						
TIME	30 | *n*	O	z			v	b
	This parameter sets the timeout value for broker calls in seconds. The results of a broker call must be received by the caller within this time limit.						
TRACE-LEVEL	0 - 4	O	z			v	b
	The level of tracing to be performed while the broker is running with transport method NET. It overrides the global value of trace level for all NET routines. 0 No tracing. Default value. 1 Display invalid Adabas commands. 2 All of trace level 1, plus errors if request entries could not be allocated. 3 All of trace level 2, plus all routines executed. 4 All of trace level 3, plus function arguments and return values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						

Security-specific Attributes

The security-specific attribute section begins with the keyword `DEFAULTS=SECURITY` as shown in the sample attribute file. This section applies only if broker-specific attribute `SECURITY=YES` is specified.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`ACCESS-SECURITY-SERVER`	NO | YES	O					b
	Determines where authentication is checked. NO Authentication is checked in the broker tasks. This requires broker to be running under TSOS in order to execute privileged security checks. YES Authentication is checked in the EntireX Broker Security Server for BS2000. This does not require broker to be running under TSOS. See *EntireX Broker Security Server for BS2000*.						
`APPLICATION-NAME`	A8	O	z				

	Specifies the name of the application to be checked if `FACILITY-CHECK=YES` is defined. In RACF, for example, an application `"BROKER"` with read permission for user `"DOE"` is defined with following commands:
	
	`RDEFINE APPL BROKER UACC(NONE)`
	`PERMIT BROKER CLASS(APPL) ID(DOE) ACCESS(READ)`
	`SETROPTS CLASSACT(APPL)`
	
	See attribute `FACILITY-CHECK` for more information.						

`AUTHORIZATION-DEFAULT`	YES | NO	O		u	w		
	Determines whether access is granted to a specified service if the specified could not be found listed in the repository of authorization rules or in section `DEFAULTS=AUTHORIZATION-RULES` of the attribute file. YES Grant access. NO Deny access. Applies only when using EntireX Security under UNIX and Windows. Authorization rules can be stored within a repository. When an authorization call occurs, EntireX Security uses the values of this parameter to perform an access check for a particular broker instance against an (authenticated) user ID and list of rules. See also *Authorization Rules*.						
`CHECK-IP-ADDRESS`	YES | NO	O	z				

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Determines whether the TCP/IP address of the caller is subject to a resource check.						
ERRTXT-MODULE	NA2MSG0 | NA2MSG1 | NA2MSG2 | *ModuleName*	O	z				
	Specifies the name of the security error text module. Default is "NA2MSG0", English messages. For instructions on how to customize messages, see *Build Language-specific Messages (Optional)* under *Installing EntireX Security under z/OS*.						
FACILITY-CHECK	NO | YES	O	z				
	It is possible to check whether a particular user is at all allowed to use an application before performing a password check. The advantage of this additional check is that when the user is not allowed to use this application, the broker returns error 00080013 and does not try to authenticate the user. Failing an authentication check may lead to the user's password being revoked; this situation is avoided if the facility check is performed first. See attribute APPLICATION-NAME for further details. **Note:** This facility check is an additional call to the security subsystem and is executed before each authentication call.						
IGNORE-STOKEN	NO | YES	O	z	u	w		b
	Determines whether the value of the ACI field SECURITY-TOKEN is verified on each call.						
INCLUDE-CLASS	YES | NO	O	z				
	Determines whether the class name is included in the resource check.						
INCLUDE-NAME	YES | NO	O	z				
	Determines whether the server name is included in the resource check.						
INCLUDE-SERVICE	YES | NO	O	z				
	Determines whether the service name is included in the resource check.						
LDAP-AUTHENTICATION-URL	*ldapUrl*	O		u	w		
	Authentication is performed against the LDAP repository specified under *ldapUrl*. ■ **TCP** Specify repository URL:						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000

	`LDAP-AUTHENTICATION-URL="ldap://HostName[:PortNumber]"`
	
	■ **SSL/TLS**
	Specify repository URL with ldaps:
	
	`LDAP-AUTHENTICATION-URL="ldaps://HostName[:PortNumber]"`
	
	If no port number is specified, the default is the standard LDAP port number 389 for TCP transport. Examples for TCP and SSL/TLS:
	
	`LDAP-AUTHENTICATION-URL="ldap://myhost.mydomain.com"`
	`LDAP-AUTHENTICATION-URL="ldaps://myhost.mydomain.com:636"`							

`LDAP-AUTHORIZATION-RULE`	*`ldapUrl`*	O		u	w		

	Authorization is performed against the LDAP repository specified under *`ldapUrl`*.
	
	■ **TCP**
	Specify repository URL:
	
	`LDAP-AUTHORIZATION-URL="ldap://HostName[:PortNumber]"`
	
	If no port number is specified, the default is the standard LDAP port number 389 for TCP transport.
	Example for TCP:
	
	`LDAP-AUTHORIZATION-URL="ldap://myhost.mydomain.com:389"`
	
	This attribute replaces the parameters `host`, `port` and `protocol` in the *xds.ini* file of EntireX version 9.10 and below.							

`LDAP-AUTH-DN`	*`authDN`*	O		u	w		
	For authenticated access to the LDAP server. Specifies the DN of the user. Default value: `cn=admin,dc=software-ag,dc=de` This attribute replaces parameter `authDN` in the *xds.ini* file of EntireX version 9.10 and below.						
`LDAP-AUTH-PASSWD-ENCRYPTED`	*`authPass`*	O		u	w		
	For authenticated access to the LDAP server. Specifies the encrypted value of the user password. Use program `etbnattr` to get the encrypted password:						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	`etbnattr -x clear_text_password -echo_password_only` This writes the encrypted password to standard output. This attribute replaces parameter `authPass` in the *xds.ini* file of EntireX version 9.10 and below.						
`LDAP-AUTHORIZATION-RULE`	A32	O		u	w		
	List of authorization rules. Multiple sets of rules can be defined, each set is limited to 32 chars. The maximum number of `LDAP-AUTHORIZATION-RULE` entries in the attribute file is 16. Applies only when using EntireX Security under UNIX or Windows and `SECURITY-SYSTEM=`*`ldapUrl`*. Authorization rules can be stored in an LDAP repository. When an authorization call occurs, EntireX Security uses the values of this parameter and `AUTHORIZATION-DEFAULT` to perform an access check for a particular broker instance against an (authenticated) user ID and list of rules. See also *Authorization Rules*.						
`LDAP-BASE-DN`	*`baseDN`*	O		u	w		
	Specifies the base distinguished name of the directory object that is the root of all objects for authorization rules. Default value: `dc=software-ag,dc=de` This attribute replaces parameter `baseDN` in the *xds.ini* file of EntireX version 9.10 and below.						
`LDAP-PERSON-BASE-BINDDN`	*`ldapDn`*	O		u	w		
	Used with LDAP authentication to specify the distinguished name where authentication information is stored. This value is prefixed with the user ID field name (see below). Example: `LDAP-PERSON-BASE-BINDDN="cn=users,dc=mydomain,dc=com"`						
`LDAP-REPOSITORY-TYPE`	OpenLDAP | ActiveDirectory | SunOneDirectory | Tivoli | Novell | ApacheDS	O		u	w		
	Use predefined known fields for the respective repository type. Specify the repository type that most closely matches your actual repository. In the case of Windows Active Directory, the user ID is typically in the form *`domainName\userId`*.						
`LDAP-SASL-AUTHENTICATION`	NO | YES	O			w		

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Specifies whether or not Simple Authentication and Security Layer (SASL) is to perform the authentication check. In practice, this determines whether or not the password supplied by the user is passed in plain text between the broker kernel and the LDAP server. If SASL is activated, this implies that the password is encrypted. NO Password is sent to LDAP server in plain text. YES Password is sent to LDAP server encrypted.						
LDAP-USERID-FIELD	cn | *uidFieldName*	O		u	w		
	Used with LDAP authentication to specify the first field name of a user in the Distinguished Name, for example: LDAP-USERID-FIELD=*uid*						
MAX-SAF-PROF-LENGTH	1-256	O	z				
	This parameter should be increased if the length of the resource checks - that is, the length of the profile comprising "<class>.<server>.<service>" - is greater than 80 bytes. This parameter defaults to 80 if a value is not specified.						
PASSWORD-TO-UPPER-CASE	NO | YES	O	z			v	
	Determines whether the password and new password are converted to uppercase before verification.						
PRODUCT	RACF | ACF2 | TOP-SECRET	O	z				
	Specifies the name of the installed security product. This attribute is used to analyze security-system-specific errors. The following systems are currently supported: ACF2 Security system ACF2 is installed. RACF Security system RACF is installed. Default. TOP-SECRET Security system TOP-SECRET is installed. The default value is used if an incorrect or no value is specified.						
PROPAGATE-TRUSTED-USERID	YES | NO	O	z				
	Determines whether a client user ID obtained by means of the trusted user ID mechanism is propagated to a server using the ACI field CLIENT-USERID.						
SAF-CLASS	NBKSAG | *SAFClassName*	O	z				
	Specifies the name of the SAF class/type used to hold the EntireX-related resource profiles.						
SAF-CLASS-IP	NBKSAG | *SAFClassName*	O	z				

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Specifies the name of the SAF class/type used when performing IP address authorization checks.						
SECURITY-LEVEL	AUTHORIZATION | AUTHENTICATION	O	z	u	w	v	b
	Specifies the mode of operation. AUTHORIZATION Authorization and authentication (not under BS2000 or z/VSE). AUTHENTICATION Authentication. **Note:** In version 8.0, the default value for this parameter was "AUTHORIZATION".						
SECURITY-NODE	YES | *name*	O	z				
	This parameter can be used to specify a prefix that is added to all authorization checks, enabling different broker kernels, in different environments, to perform separate authorization checks according to each broker kernel. For example, it is often important to distinguish between production, test, and development environments. YES This causes the broker ID to be used as a prefix for all authorization checks. *name* This causes the actual text (maximum 8 characters) to be prefixed onto all authorization checks. **Note:** By *not* setting this parameter, no prefix is added to the resource check (the default behavior).						
SECURITY-SYSTEM	OS | LDAP	O	z	u	w		b
	OS Authentication is performed against the local operating system. Default if SECURITY=YES is specified and section DEFAULTS=SECURITY is omitted from the attribute file. LDAP Authentication and authorization are performed against the LDAP repository specified under LDAP-AUTHENTICATION-URL and LDAP-AUTHORIZATION-URL.						
TRACE-LEVEL	0 - 4	O	z	u	w	v	b
	Trace level for EntireX Security. It overrides the global value of trace level in the attribute file. 0 No tracing. Default value. 1 Log security violations and access denied/permitted. 2 All of trace level 1, plus internal errors. 3 All of trace level 2, plus function entered/exit messages with argument values and some progress messages.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	4 All of trace level 3, plus some selected data areas for problem analysis. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD. **Note:** Setting this value also affects tracing for authorization rules.						
TRUSTED-USERID	YES | NO	O	z				
	Activates the trusted user ID mechanism for broker requests arriving over the local Adabas IPC mechanism.						
USERID-TO-UPPER-CASE	NO | YES	O	z			v	
	Determines whether user ID is converted to uppercase before verification.						
UNIVERSAL	NO | YES	O	z				
	Determines whether access to undefined resource profiles is allowed.						
WARN-MODE	NO | YES	O	z	u	w		b
	Determines whether a resource check failure results in just a warning or an error.						

TCP/IP-specific Attributes

The TCP/IP-specific attribute section begins with the keyword `DEFAULTS=TCP` as shown in the sample attribute file. It contains attributes that apply to the TCP/IP transport communicator. The transport is activated by `TRANSPORT=TCP` in the Broker-specific section of the attribute file. A maximum of five TCP/IP communicators can be activated by specifying up to five `HOST/PORT` pairs.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`CONNECTION-NONACT`	*n* | *n*S | *n*M | *n*H	O	z	u	w	v	b
	Non-activity of the TCP/IP connection, after which a close is performed and the connection resources are freed. If this parameter is not specified, broker will close the connection only when the application (or the network itself) terminates the connection. *n* Same as *n*S. *n*S Non-activity time in seconds (min. 600, max. 2147483647). *n*M Non-activity time in minutes (min. 10, max. 35791394). *n*H Non-activity time in hours (max. 596523). If not specified, the connection non-activity test is disabled. On the stub side, non-activity can be set with the environment variable `ETB_NONACT`. See *Limiting the TCP/IP Connection Lifetime* in the platform-specific *Stub Administration* sections of the EntireX documentation.						
`HOST`	0.0.0.0 | *HostName* | *IP address*	O	z	u	w	v	b
	The address of the network interface on which broker will listen for connection requests. If `HOST` is not specified, broker will listen on any attached interface adapter of the system (or stack). A maximum of five `HOST/PORT` pairs can be specified to start multiple instances of broker's TCP/IP transport communicator.						
`MAX-MESSAGE-LENGTH`	2147483647 | *n*	O	z	u	w	v	b

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Maximum message size that the broker kernel can process using transport method TCP/IP. The default value represents the highest positive number that can be stored in a four-byte integer.						
PORT	1025 - 65535	O	z	u	w	v	b
	The TCP/IP port number on which the broker will listen for connection requests. If not specified, the broker will attempt to find its TCP/IP port number from the TCP/IP services file, using `getservbyname`. If it cannot find the number here, the default value of 1971 is used. A maximum of five `HOST/PORT` pairs can be specified to start multiple instances of broker's TCP/IP transport communicator.						
RESTART	YES | NO	O	z	u	w	v	b
	YES The broker kernel will attempt to restart the TCP/IP communicator. NO The broker kernel will not try to restart the TCP/IP communicator. This setting applies to all TCP/IP communicators.						
RETRY-LIMIT	20 | *n* | UNLIM	O	z	u	w	v	b
	Maximum number of attempts to restart the TCP/IP communicator. This setting applies to all TCP/IP communicators.						
RETRY-TIME	3M | *n* | *n*S | *n*M | *n*H	O	z	u	w	v	b
	Wait time between stopping the TCP/IP communicator due to an unrecoverable error and the next attempt to restart it. *n* Same as *n* S. *n* S Wait time in seconds (max. 2147483647). *n* M Wait time in minutes (max. 35791394). *n* H Wait time in hours (max. 596523). Minimum wait time is 1S. This setting applies to all TCP/IP communicators.						
REUSE-ADDRESS	YES | NO	O	z	u		v	b
	YES | NO	O			w		

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	YES The TCP port assigned to the broker can be taken over and assigned to other applications (this is the default value on all non-Windows platforms). NO The TCP port assigned to the broker cannot be taken over and assigned to other applications. This is the default setting on Windows, and we strongly advise you do not change this value on this platform. **Note:** This setting might be required at your site when restarting broker immediately after stopping it. This is due to the inherent latency of the TCP/IP stack when closing connections.						
STACK-NAME	*StackName*	O	z				
	Name of the TCP/IP stack that the broker is using. If not specified, broker will connect to the default TCP/IP stack running on the machine.						
TRACE-LEVEL	0 - 4	O	z	u	w	v	b
	The level of tracing to be performed while the broker is running with transport method TCP/IP. It overrides the global value of trace level for all TCP/IP routines. 0 No tracing. Default value. 1 Display IP address of incoming request, display error number of outgoing error responses. 2 All of trace level 1, plus errors if request entries could not be allocated. 3 All of trace level 2, plus all routines executed. 4 All of trace level 3, plus function arguments and return values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						

c-tree-specific Attributes

The c-tree-specific attribute section begins with the keyword `DEFAULTS = CTREE`. The attributes in this section are optional. This section applies only if `PSTORE-TYPE = CTREE` is specified.

Not available under z/OS, BS2000, z/VSE.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`MAXSIZE`	*n* | *n*M | *n*G	O		u	w		
	Defines the maximum size of c-tree data files. Broker allocates one data file for control data and another data file for message data: *n* Maximum size in MB. *n*M Maximum size in MB. *n*G Maximum size in GB.						
`PAGESIZE`	*n* | *n*K	O		u	w		
	Determines how many bytes are available in each c-tree node. `PSTORE COLD` start is required after changing this value. *n* Same as *n*K *n*K PAGESIZE in KB. The default and minimum value is 8 KB. If `PSD Reason Code = 527` is returned during UOW write processing, increase the `PAGESIZE` value and restart broker with `PSTORE=COLD`, or migrate the existing PSTORE to a new PSTORE with an increased `PAGESIZE` value. See *Migrating the Persistent Store* and define the increased `PAGESIZE` value for the load broker.						
`PATH`	A255	O		u	w		
	Path name of the target directory for c-tree index and data files.						
`SYNCIO`	NO | YES	O		u	w		
	Controls the open mode of the c-tree transaction log. NO c-tree transaction log is not opened in synchronous mode. Default. YES c-tree transaction log is opened in synchronous mode to improve data security. It may degrade performance of PSTORE operations, but offers the highest level of data security. See *c-tree Database as Persistent Store* in the UNIX and Windows Administration documentation.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
TRACE-LEVEL	0 - 4	O		u	w		
	Trace level for c-tree persistent store. It overrides the global value of trace level in the attribute file. 0 No tracing. Default value. 1 Log memory allocation failures and errors during close of files. 2 n/a 3 All of trace level 1, plus UOWID in use for the various ctree requests and function entered/exit mesages. 4 All of trace level 3, plus returned function values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						

SSL/TLS-specific Attributes

The Broker can use Secure Sockets Layer/Transport Layer Security (SSL/TLS) as the transport medium. The term “SSL” in this section refers to both SSL and TLS. RPC-based clients and servers, as well as ACI clients and servers, are always SSL clients. The broker is always the SSL server. For an introduction see *SSL/TLS and Certificates with EntireX*.

The SSL-specific attribute section begins with the keyword `DEFAULTS=SSL` as shown in the sample attribute file. The attributes in this section are needed to execute the SSL communicator of the EntireX Broker kernel.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`CIPHER-SUITE`	*`string`*	O	z	u	w		b

	String that is passed to the underlying SSL/TLS implementation. SSL/TLS is a standardized protocol that uses different cryptographic functions (hash functions, symmetric and asymmetric encryption etc.). Some of these must be implemented in the SSL/TLS stack; others are optional. When an SSL/TLS connection is created, both parties agree by "handshake" on the *`cipher suite`*, that is, the algorithms and key lengths used. In a default scenario, this information depends on what both sides are capable of. It can be influenced by setting the attribute `CIPHER-SUITE` for the SSL/TLS server side (the broker always implements the server side). Thus stubs connect to the broker and thereby become the SSL/TLS clients.
	
	Under UNIX, Windows and BS2000, the OpenSSL implementation is used; under z/OS it is GSK.
	
	The SSL protocol is obsolete and should no longer be used for secure operations. The TLS protocol is the successor of SSL and is readily available in OpenSSL and GSK. The following examples show how to configure strong encryption:
	
	■ **OpenSSL**
	This example uses FIPS-approved algorithms, but without ADH, MD5 or other low or export-grade implementations.
	It requires authentication and encryption, that is, do not use NULL ciphers:
	
	`CIPHER-SUITE=FIPS:!ADH:!LOW:!EXP:!MD5:!aNULL:!eNULL:@STRENGTH`
	
	Default configuration:
	
	`CIPHER-SUITE=TLSv1:!ADH:!LOW:!EXP: !MD5:@STRENGTH`
	
	See *http://www.openssl.org/docs/apps/ciphers.html*						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	■ **GSK** Default configuration: `CIPHER-SUITE=35363738392F303132330A1613100D` This list of cipher suites starts with a strong '256-bit AES encryption with SHA-1 message authentication and RSA key exchange' (35) and ends with a relatively weak '168-bit Triple DES encryption with SHA-1 message authentication and fixed Diffie-Hellman key exchange signed with a DSA certificate' (0D). See IBM documentation: z/OS V2R1.0 System SSL Programming, SC14-7495-00, Appendix C: Cipher Suite Definitions.							
`CONNECTION-NONACT`	*n* | *n*S | *n*M | *n*H	O	z	u	w		b
	Non-activity of the SSL connection, after which a close is performed and the connection resources are freed. If this parameter is not specified here, broker will close the connection only when the application (or the network itself) terminates the connection. *n* Same as *n*S. *n*S Non-activity time in seconds (min. 600, max. 2147483647). *n*M Non-activity time in minutes (min. 10, max. 35791394). *n*H Non-activity time in hours (max. 596523). If not specified, the connection non-activity test is disabled.						
`HOST`	*hostname*	O	z	u	w		b
	The address of the network interface on which broker will listen for connection requests. If `HOST` is not specified, broker will listen on any attached interface adapter of the system (or stack). A maximum of five `HOST/PORT` pairs can be specified to start multiple instances of EntireX Broker's TCP/IP transport communicator.						
`KEY-LABEL`	*name*	O	z				
	The label of the key in the RACF keyring that is used to authenticate the broker kernel (see also `TRUST-STORE` parameter). Example: "ETBCERT".						
`KEY-FILE`	*file name*	R		u	w		b
	File that contains the broker's private key (if not contained in `KEY-STORE`). For test purposes, EntireX delivers certificates for use on various platforms. See *SSL/TLS Sample Certificates Delivered with EntireX*.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	Example for UNIX and Windows: *MyAppKey.pem*. **Note:** EntireX Broker does not support Java certificates (keystore files of type .jks).						
KEY-PASSWD	*password* (A32)	R		u	w		b
	Password used to protect the private key. Unlocks the KEY-FILE, for example *MyAppKey.pem*. Deprecated. See KEY-PASSWD-ENCRYTPED below.						
KEY-PASSWD-ENCRYPTED	*encrypted value* (A64)	R		u	w		b
	Password used to protect the private key. Unlocks the KEY-FILE, for example *MyAppKey.pem*. This attribute replaces KEY-PASSWD to avoid a clear-text password as attribute value. If KEY-PASSWD and KEY-PASSWD-ENCRYTPED are both supplied, KEY-PASSWD-ENCRYTPED takes precedence.						
KEY-STORE	*file name*	R		u	w		b
	SSL certificate; may contain the private key. For test purposes, EntireX delivers certificates for use on various platforms. See *SSL/TLS Sample Certificates Delivered with EntireX*. Example for UNIX and Windows: *ExxAppCert.pem*. **Note:** EntireX Broker does not support Java certificates (keystore files of type .jks).						
MAX-MESSAGE-LENGTH	2147483647 | *n*	O	z	u	w		b
	Maximum message size that the broker kernel can process using transport method SSL. The default value represents the highest positive number that can be stored in a four-byte integer.						
PORT	*1025 - 65535*	O	z	u	w		b
	The SSL port number on which the broker will listen for connection requests. If not changed, this parameter takes the standard value as specified in the example attribute file. If the port number is not specified, the broker will use the default value of 1958.						
RESTART	YES | NO	O	z	u	w		b
	YES The broker kernel will attempt to restart the SSL communicator (this is the default value). NO The broker kernel will not attempt to restart the SSL communicator.						
RETRY-LIMIT	20 | *n* | UNLIM	O	z	u	w		b
	Maximum number of attempts to restart the SSL communicator.						
RETRY-TIME	3M | *n* | *n*S | *n*H	O	z	u	w		b
	Wait time between suspending SSL communication due to unrecoverable error and the next attempt to restart it.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	n Same as *n*S. *n*S Wait time in seconds (max.2147483647). *n*M Wait time in minutes (max. 35791394). *n*H Wait time in hours (max. 596523). Minimum: 1S						
REUSE-ADDRESS	YES | NO	O	z	u	w		b
	YES The SSL port assigned to the broker can be taken over and assigned to other applications (this is the default value). NO The SSL port assigned to the broker cannot be taken over and assigned to other applications. **Note:** This setting might be required at your site when restarting broker immediately after stopping it. This is due to the inherent latency of the TCP/IP stack when closing connections.						
STACK-NAME	*name*	O	z	u	w		
	Name of the TCP/IP stack that the broker is using. If not specified, broker will connect to the default TCP/IP stack running on the machine.						
TRACE-LEVEL	0 - 4	O	z	u	w		b
	The level of tracing to be performed while the broker is running with transport method SSL/TLS. It overrides the global value of trace level for all SSL/TLS routines. 0 No tracing. Default value. 1 Display IP address of incoming request, display error number of outgoing error responses. 2 All of trace level 1, plus errors if request entries could not be allocated. 3 All of trace level 2, plus all routines executed. 4 All of trace level 3, plus function arguments and return values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						
TRUST-STORE	*file name|keyring*	R	z	u	w		b
	Location of the store containing certificates of trust Certificate Authorities (or CAs).						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	z/OS Specify the RACF keyring using the following format: `[USER-ID/]RING-NAME`. If no value for `USER-ID` is provided, the keyring is assumed to be associated with the user ID that the broker kernel is running under. BS2000/Windows/UNIX Specify the file name of the CA certificate store. Examples: *EXXCACERT.PEM, C:\Certs\ExxCACert.pem*						
`VERIFY-CLIENT`	NO | YES	O	z	u	w		b
	YES Additional client certificate required. NO No client certificate required (default).						

DIV-specific Attributes

These attributes define a persistent store that is implemented as a VSAM linear data set (LDS) accessed using Data In Virtual (DIV). This DIV persistent store is a container for units of work. The DIV-specific attribute section begins with the keyword `DEFAULTS = DIV`. The attributes in this section are required if `PSTORE-TYPE = DIV` is specified.

Note: All attributes except the deprecated `DIV` were introduced with EntireX version 9.12. They replace the *Format Parameters* of earlier versions, which are deprecated but still supported for compatibility reasons.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`DIV`	A511	O	z				
	The VSAM persistent store parameters, enclosed in double quotes (""). The value can span more than one line. **Note:** Deprecated. This attribute is applicable only if you are supplying the persistent store parameters using *Format Parameters* of earlier versions. We recommend you use the attributes below that were introduced with EntireX 9.12 instead.						
`DATASPACE-NAME`	A8	O	z				
	Defines the name of the dataspace that will be used to map the persistent store. Default value is `DSPSTORE`.						
`DATASPACE-PAGES`	126-524284	O	z				
	Defines the size of the dataspace used to map the persistent store (size=`DATASPACE-PAGES` * 4 KB). We recommend using the maximum value. Default value is 2048.						
`DDNAME`	A8	R	z				
	Defines the JCL `DDNAME` that will be used to access the persistent store.						
`STORE`	A8	R	z				
	Defines an internal name that is used to identify the persistent store.						
`TRACE-LEVEL`	0 - 4	O	z				
	Trace level for DIV. It overrides the global value of trace level in the attribute file. 0 No tracing. Default value. 1 Log selected `DIV SAVE` calls taking longer than 2 seconds elapsed time. 2 n/a						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
	3 All of trace level 1, plus UOWID in use for the various DIV requests. 4 n/a Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						

Adabas-specific Attributes

The Adabas-specific attribute section begins with the keyword DEFAULTS = ADABAS. The attributes in this section are required if PSTORE-TYPE = ADABAS is specified. In previous versions of EntireX, these Adabas-specific attributes and values were specified in the broker-specific PSTORE-TYPE attribute.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
BLKSIZE	126-20000	O	z	u	w	v	b
	Optional blocking factor used for message data. If not specified, broker will split the message data into 2 KB blocks to be stored in Adabas records. The maximum value depends on the physical device assigned to data storage. See the *Adabas* documentation. For reasons of efficiency, do not specify a BLKSIZE much larger than the actual total size of the UOW data to be written. The total UOW size is the sum of all messages in the UOW plus 41 bytes of header information. This takes effect only after COLD start. The BLKSIZE parameter applies only for a cold start of broker; subsequently the value of BLKSIZE is taken from the last cold start. Default value is 2000.						
DBID	1 - 32535	R	z	u	w	v	b
	Database ID of Adabas database where the persistent store resides.						
FNR	1 - 32535	R	z	u	w	v	b
	File number of broker persistent store file.						

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
FORCE-COLD	N | Y	O	z	u	w	v	b
	Determines whether a broker cold start is permitted to overwrite a persistent store file that has been used by another broker ID and/or platform. Specify Y to allow existing information to be overwritten.						
MAXSCAN	0-*n*	O	z	u	w	v	b
	Limits display of persistent UOW information in the persistent store through Command and Information Services. Default value is 1000.						
OPENRQ	N | Y	O	z	u	w	v	b
	Determines whether driver for Adabas persistent store is to issue an OPEN command to Adabas.						
SVC	200-255	R	z			v	
	Use this parameter to specify the Adabas SVC number to be used by the Adabas persistent store driver.						
TRACE-LEVEL	0 - 4	O	z	u	w	v	b
	Trace level for Adabas persistent store. It overrides the global value of trace level in the attribute file. 0 No tracing. Default value. 1 Log selected Adabas CB fields (command code, response code, subcode, ISN, additions). 2 n/a 3 All of trace level 1, plus UOWID in use for the various Adabas requests and function entered/exit mesages. 4 All of trace level 3, plus more Adabas CB fields for successful requests and returned function values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. For temporary changes to TRACE-LEVEL without a broker restart, use the EntireX Broker command-line utility ETBCMD.						

Application Monitoring-specific Attributes

The application monitoring-specific attribute section begins with the keyword DEFAULTS=APPLICATION-MONITORING. It contains attributes that apply to the application monitoring functionality. At startup time, the attributes are read if the Broker-specific attribute APPLICATION-MONITORING=YES is specified. Duplicate or missing values are treated as errors. When an error occurs, application monitoring is turned off and EntireX Broker continues execution. See *Application Monitoring*.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
APPLICATION-MONITORING-NAME or APPMON-NAME	A100	O	z	u	w	v	
	Specifies a default application monitoring name. Used to set the value of the **ApplicationName** KPI.						
COLLECTOR-BROKER-ID	A64	R	z	u	w	v	
	Identifies the Application Monitoring Data Collector. Has the format *host_name*:*port_number*, where *host_name* is the host where the Application Monitoring Data Collector is running and *port_number* is the port number of the Application Monitoring Data Collector. The default port is 57900.						
TRACE-LEVEL	0 - 4	O	z	u	w	v	
	The level of tracing to be performed while the broker is running with application monitoring. 0 No tracing. Default value. 1 Display application monitoring errors. 2 All of trace level 1, plus measuring points for application monitoring. 3 All of trace level 2, plus function entered/exit messages with argument values and monitoring buffers. 4 All of trace level 3, plus returned function values. Trace levels 2, 3 and 4 should be used only when requested by Software AG support. If you modify the TRACE-LEVEL attribute, you must restart the broker for the change to take effect. TRACE-LEVEL cannot be changed dynamically for application monitoring.						

Authorization Rule-specific Attributes

The authorization rule-specific attribute section begins with the keyword `DEFAULTS=AUTHORIZATION-RULES`. It contains attributes that enhance security-related definitions. At startup time, the attributes are read if the following conditions are met:

- Broker-specific attribute `SECURITY=YES`
- Security-specific attributes `SECURITY-SYSTEM=OS` and `SECURITY-LEVEL=AUTHORIZATION`

When an error occurs, the EntireX Broker stops. See *Authorization Rules*.

Attribute	Values	Opt/ Req	Operating System				
			z/OS	UNIX	Windows	z/VSE	BS2000
`RULE-NAME`	A32	R		u	w		
	Specifies a rule name. A rule is a container for a list of services and a list of client and server user IDs. All users defined in a rule are authorized to use all services defined in this rule. See example under *Rules Stored in Broker Attribute File*.						

`CLASS`
`SERVER`
`SERVICE`
	A32	R		u	w		

	These three attributes together identify the service. `CLASS` must be specified first, followed immediately by `SERVER` and `SERVICE`. *Wildcard Service Definitions* under *Broker Attributes* are allowed.						
`CLIENT-USER-ID`	A32	R		u	w		
	Defines an authorized client user ID.						
`SERVER-USER-ID`	A32	R		u	w		
	Defines an authorized server user ID.						

Variable Definition File

The broker attribute file contains the configuration of one EntireX Broker instance. In order to share attribute files between different brokers, you identify the attributes that are unique and move them to a variable definition file. This file enables you to share one attribute file among different brokers. Each broker in such a scenario requires its own variable definition file.

The following attributes are considered unique for each machine:

- `BROKER-ID` (in Broker-specific Attributes)
- `NODE` (in Adabas SVC/Entire Net-Work-specific Attributes)
- `PORT` (in SSL/TLS-specific Attributes and TCP/IP-specific Attributes)

How you use the variable definition file will depend upon your particular needs. For instance, some optional attributes may require uniqueness - for example, `DBID` and `FNR` in `DEFAULTS=ADABAS` - so that you may specify the persistent store.

III Broker Command and Information Services

8 Broker Command and Information Services

EntireX Broker provides two internal services: Command Service and Information Services that can be used administer and monitor the EntireX Broker. The command service allows you to issue a set of Broker commands; the information services provide you with various statistics to better administer and tune your Broker. Because these services are implemented internally, nothing has to be started or configured. You can use these services immediately after starting EntireX Broker.

See also *Broker CIS Data Structures*.

CIS Overview Table

EntireX Broker provides these predefined internal services:

- **Command Service**
  Provides a facility to issue commands against the Broker (e.g. `SHUTDOWN` etc.).
- **Information Services**
  Provides a query mechanism to obtain various types of information on the Broker, which is helpful for administration and tuning.

Since these services are implemented internally, nothing has to be started, configured or defined in the Broker attribute file. You can use them immediately after starting the Broker. They can be requested as follows:

Mode of Request	Tools	Services	Requirements
User-Written Interface	application program	■ `INFO` ■ `USER-INFO` ■ `CMD` ■ `PARTICIPANT-SHUTDOWN` ■ `SECURITY-CMD`	■ request structures
Command-line Utilities	`ETBINFO` utility	■ `INFO` ■ `USER-INFO`	■ profile ■ command-line parameters
	`ETBCMD` utility	■ `CMD` ■ `PARTICIPANT-SHUTDOWN` ■ `SECURITY-CMD`	■ command-line parameters

Applicable operating systems: z/OS, UNIX, Windows and z/VSE.

Description of Services

INFO and USER-INFO

- `INFO` is the full information service. Specify it for the full information service. All clients, servers and conversations are listed.
- `USER-INFO` is limited to your user-specific information. Specify it for limited information service. Only the user's own resources are listed.

CMD, PARTICIPANT-SHUTDOWN and SECURITY

- `CMD` is the full command service.
- `PARTICIPANT-SHUTDOWN` is limited to shutting down participants.
- `SECURITY-CMD` is limited to EntireX Security-related commands.

Modes of Requesting the Services

Use one of these three modes to request a service:

- Command-line Utilities
- Graphical User Interface
- User-Written Interface

The method for requesting these services is the same as the method for requesting any other service. For both types of services, an application issues a `SEND` command with appropriate data and retrieves a reply. The request itself is specified within the `SEND` buffer; the reply - if there is one - is specified in the `RECEIVE` buffer.

For Information Services requests, `RECEIVE` operations must be repeated until the Information Service indicates the end of data with an `EOC` return message.

Command-line Utilities

Software AG provides three command-line utility programs for use with EntireX Broker. All utility programs use command-line parameters that specify various options and information to be built into a request. These utility programs are:

- `ETBINFO`
  Queries the Broker for different types of information, generating an output text string with basic formatting. This text output can be further processed by script languages (or elsewhere). `ETBINFO` uses data descriptions called profiles to control the type of data that is returned for a request. `ETBINFO` is useful for configuring and administering EntireX Broker efficiently - e.g., how many

users are to run concurrently and whether the number of specified message containers is large enough.

See `ETBINFO` under *Broker Command-line Utilities* in the platform-specific Administration documentation for profiles, examples and utility parameters.

- `ETBCMD`
  Allows you to take actions - e.g., purge a unit of work, stop a server, shut down a Broker - against EntireX Broker.

  See `ETBCMD` under *Broker Command-line Utilities* in the platform-specific Administration documentation for utility parameters.

Version Information

- The `ETBINFO` and `ETBCMD` CIS command-line utilities are compatible with all versions of EntireX Broker.
- Display keywords applying to a specific version of Broker will not be returned when a call is made to any older version of Broker.

Graphical User Interface

Software AG provides a graphical user interface, Command Central, for displaying information on the Broker and/or executing administrative functions.

Software AG Command Central is a tool that release managers, infrastructure engineers, system administrators, and operators can use to perform administrative tasks from a single location. Command Central can assist with the following configuration, management, and monitoring tasks:

- Infrastructure engineers can see at a glance which products and fixes are installed, where they are installed, and compare installations to find discrepancies.
- System administrators can configure environments by using a single web user interface or command-line tool. Maintenance involves minimum effort and risk.
- Release managers can prepare and deploy changes to multiple servers using command-line scripting for simpler, safer lifecycle management.
- Operators can monitor server status and health, as well as start and stop servers from a single location. They can also configure alerts to be sent to them in case of unplanned outages.

User-Written Interface

If you access the Command and Information Services through a user-written application, you must use a defined protocol. This protocol describes the structures needed to communicate with the service(s) so that the request is correctly interpreted by the Broker.

See Writing Applications: Command and Information Services and *Broker CIS Data Structures*.

ETBCMD: Executable Command Requests

The following command requests can be issued, using `ETBCMD`. All the functions listed in this table are applicable to all three request modes; see ***Modes of Requesting the Services***.

Note: Version numbers in this table refer to the interface version and not to the Broker version.

Command Request	Comment	CIS Interface Version
`ALLOW-NEWUOWMSGS`	New UOW messages are allowed.	3
`CLEAR-CMDLOG-FILTER`	Remove the specified command log filter.	5
`CONNECT-PSTORE`	Connects the persistent store. See *Availability of Persistent Store*.	4
`DISABLE-ACCOUNTING`	Disables accounting. Accounting records are discarded until accounting is enabled.	5
`DISABLE-CMDLOG`	Disable command logging.	5
`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.	7
`DISCONNECT-PSTORE`	Disconnects the persistent store. See *Availability of Persistent Store*.	4
`ENABLE-ACCOUNTING`	Enable accounting.	5
`ENABLE-CMDLOG`	Enable command logging.	5
`ENABLE-DYN-WORKER`	Enable the `DYNAMIC-WORKER-MANAGEMENT` again.	

Command Request			Comment	CIS Interface Version
			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.	
FORBID-NEWUOWMSGS			New UOW messages are not allowed.	3
PRODUCE-STATISTICS			Output current statistics to the broker log.	5
PURGE			Remove a unit of work from the persistent store.	2
RESET-USER			Clear all cached security information for the specified user ID.	5
RESUME			Transport ID: NET|S*nn*|T*nn*. Resume a suspended transport communicator. If the communicator was not suspended before, an error message will be returned.	
SET-CMDLOG-FILTER			Add the specified command log filter.	5
SHUTDOWN	BROKER		Shutdown Broker immediately.	1
	CONVERSATION <*conversation-id*>	Command applies to conversations without units of work only. The security rights shutting down the service are required for shutting down the conversation.		7
		IMMED	The specified conversation is immediately removed. All messages of the conversation are lost.	
		QUIESCE	An end of conversation is issued. The conversation remains active.	
	SERVER	IMMED	Shutdown server immediately. The server must be uniquely identified using field P-USER-ID or SEQNO and will be completely removed from the broker environment. The following steps will be performed: ■ Error code 00100050 will be returned to the server, if it is waiting. ■ All existing conversations will be finished with EOC. ■ User will be logged off.	1

Command Request			Comment	CIS Interface Version
		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).	
	SERVICE <*class/server/service*>	Internal services cannot be shut down.		7
		IMMED	Caution: All servers offering this service will be deregistered and logged off. The following steps will be performed: ■ Error code 00100050 will be replied to all servers, if they are waiting. ■ All existing conversations will be finished with EOC. ■ Users will be logged off.	
		QUIESCE	All servers offering this service are deregistered. Shutdown servers but allow existing conversations to continue. The termination is signaled to the servers by error code 00100051. After this, the next call issued must be a DEREGISTER for the service.	
	PARTICIPANT	IMMED	Shutdown participant immediately. The participant must be identified, using fields P-USER-ID, UID TOKEN or SEQNO and will be completely removed from the Broker environment. See *Broker CIS Data Structures*. 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. Within EntireX Broker nomenclature, a participant is an	4

Command Request			Comment	CIS Interface Version
			application implicitly or explicitly logged on to the Broker as a specific user. A participant could act as client or server.	
		QUIESCE	Shutdown participant but allow existing conversations to continue. The termination is signaled to the participant by error code 00100051.	
START	TRANSPORT	Transport ID: NET|S*nn*|T*nn*	Start a transport communicator that was previously stopped. If the communicator was not stopped before, an error message will be returned.	7
STATUS	TRANSPORT	Transport ID: NET|S*nn*|T*nn*	Check the current status of the transport communicator.	7
STOP	TRANSPORT	Transport ID: NET|S*nn*|T*nn*	Stop an active or suspended transport communicator. The transport communicator will shut down. All transport-specific resources will be freed. User requests receive response code 148.	7
SUSPEND	TRANSPORT	Transport ID: NET|S*nn*|T*nn*	Suspend an active transport communicator.	7
SWITCH-CMDLOG			Force a switch of command logging output files.	5
TRACE-FLUSH	BROKER		Flush all trace data kept in internal trace buffers to stderr (DD:SYSOUT). The broker-specific attribute TRMODE=WRAP is required.	7
TRACE-OFF	BROKER		Set TRACE-LEVEL off in Broker.	1
	PSF		Set TRACE-LEVEL off in persistent store.	5
	SECURITY		Set TRACE-LEVEL off in EntireX Security.	5
TRACE-ON	BROKER		Set TRACE-LEVEL on in Broker. Values: 1 | 2 | 3 | 4.	1
	PSF		Set TRACE-LEVEL on in persistent store. Values: 1 | 2 | 3 | 4.	5
	SECURITY		Set TRACE-LEVEL on in EntireX Security. Values: 1 | 2 | 3 | 4.	5

Command Request			Comment	CIS Interface Version
TRAP-ERROR	BROKER	Error number: *nnnn*	Modifies the setting of the broker-specific attribute TRAP-ERROR.	7

ETBINFO: Returnable Information Requests

The following information requests can be returned. All the functions listed in this table are applicable to all three request modes; see *Modes of Requesting the Services*.

Note: Version numbers in this table refer to the interface version and not to the Broker version.

Information Request	Comment	Interface Version
BROKER	Global information on this Broker. No additional selection criteria are needed. Other selection criteria fields are ignored.	1
CLIENT	Information on active clients.	1
CMDLOG-FILTER	Information on command log filters.	5
CONVERSATION	Information on active conversations.	1
NET	Information on the Entire Net-Work communicator.	5
POOL	Information on Broker pool usage and dynamic memory management.	7
PSF	Information on a unit of work's status and Information for persistent store.	2
PSFDIV	Global information on the DIV persistent store.	2
PSFADA	Global information on the Adabas persistent store.	3
PSFCTREE	Global information on the c-tree persistent store.	5
RESOURCE	Information on Broker resource usage.	7
SECURITY	Global information on EntireX Security.	5
SERVER	Information on active servers.	1
SERVICE	Information on active services.	1
SSL	Information on the SSL communicator.	5
STATISTICS	Statistics on selected Broker resources.	7
TCP	Information on the TCP/IP communicator.	5
UOW-STATISTICS	Statistics on UOWs of selected services.	9
USER	Information on all users of Broker regardless of the user type.	7
WORKER	Global information on all workers. No additional selection criteria are needed. Other selection criteria fields are ignored.	1
WORKER_USAGE	Information on usage of worker tasks and dynamic worker management.	7

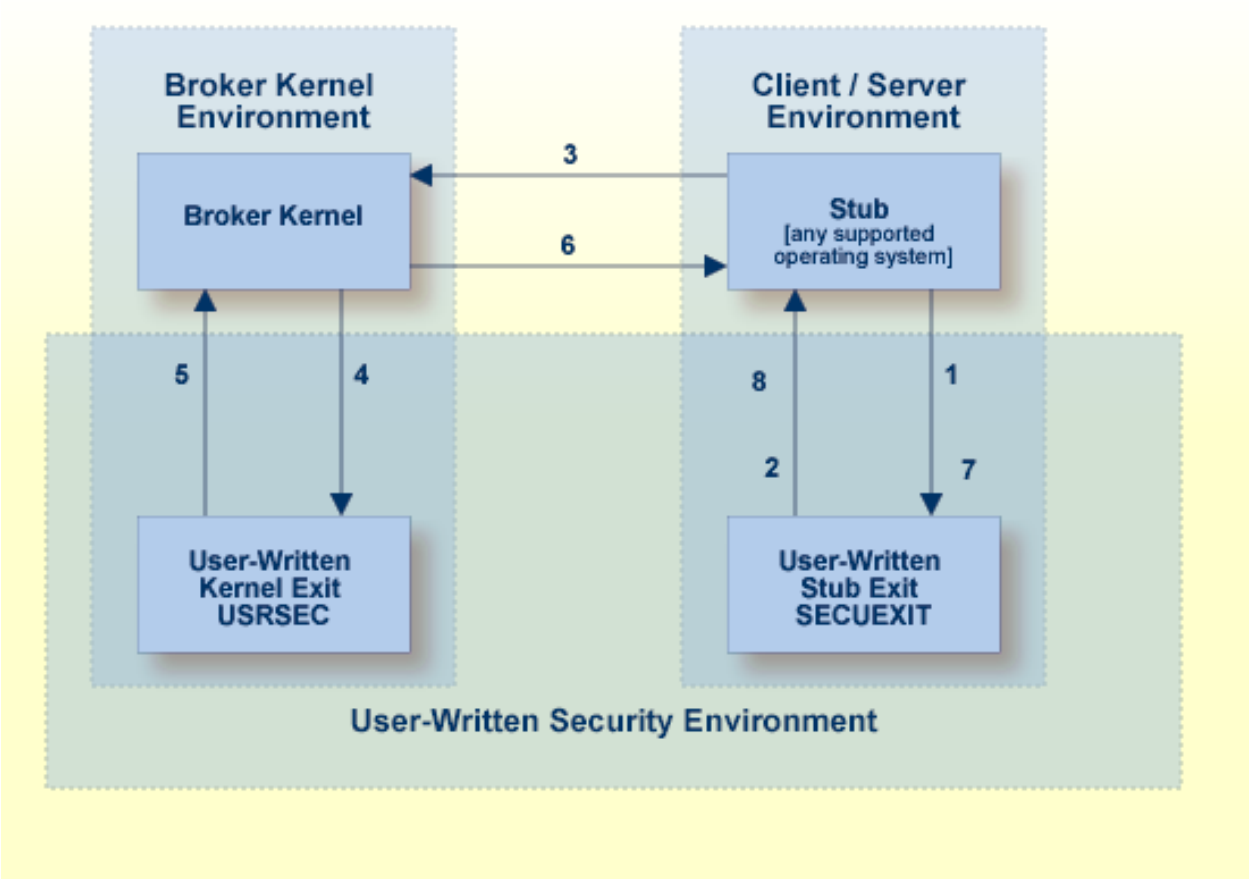
IV Using Sample Security Exits for Broker Security

9 Using Sample Security Exits for Broker Security

This page describes implementation issues and how to use sample security exits in EntireX Broker. It assumes you are familiar with EntireX Broker from both an administrative and an application perspective, and with the ACI programming interface in particular. See *Introduction to ACI-based Programming*.

Overview of Security Data Flow

The diagram shows a data flow for sample security exits, with Broker Kernel located, for example, on z/OS. See also *Description of Steps in Data Flow*.



Prerequisites for Running EntireX Broker in a Secure Environment

To run EntireX Broker in a secure environment, the following prerequisites must be met:

- The security system in the EntireX Broker kernel must be activated by setting `SECURITY=YES` in the broker attribute file.
- The security routines must be accessible to the Broker. The method you use to achieve this depends on the operating system where your user-written USRSEC is implemented.

Note: EntireX Broker will not start if `SECURITY=YES` is specified but the security routines cannot be activated.

General Security Recommendations

If you run a secure environment, we recommend you performing an explicit LOGON with the `AUTOLOGON=NO` definition in the attribute file. All security violations are logged to the EntireX Broker log file.

- Implementing Security Exits under z/OS
- Implementing Security Exits under UNIX
- Implementing Security Exits under Windows

Implementing Security Exits under z/OS

Broker Kernel

To implement the kernel security exit under z/OS

1. Write the exits `USRSEC`. The code must always be reentrant and reusable.
2. The kernel security exit `USRSEC` is loaded automatically during startup of Broker. Use module and entry name `USRSEC` for this exit. A security module sample source is delivered with the ETB source library.
3. Under z/OS, link the exit as reentrant and reusable.
4. Ensure that the security exit is accessible in the Broker `STEPLIB`.

Broker Stubs

To implement security exits for Broker stubs under z/OS

1 Write the stub security exits `ETBUPRE` and `ETBUEVA`. The code must always be reentrant, except for batch, where the code must be reusable.

2 Link these exits `ETBUPRE` and `ETBUEVA` to the stub of the target application. The stub contains weak externals for both entries.

Implementing Security Exits under UNIX

Broker Kernel

To implement the kernel security exit under UNIX

1 Write your own *usrsec.c* based on the samples delivered with EntireX.

2 Build your own *usrsec.s[o|l]*, using the provided makefiles. (A sample makefile, *makexa*, is provided.).

3 Ensure that *usrsec.s[o|l]* is made available to the Broker kernel at execution time. The attribute file parameter `SECURITY-PATH` must be used to specify the location of *usrsec.s[o|l]*.

Broker Stubs

To implement security exits for Broker stubs under UNIX

1 Write your own *secuexit.c*, based on the samples delivered with EntireX.

2 Build your own *secuexit.s[o|l]*, using the provided makefiles. (A sample makefile, *makexa*, is provided.).

3 Ensure that *secuexit.s[o|l]* is made available to the application in the same directory as the Broker stub.

Implementing Security Exits under Windows

Broker Kernel

To implement the kernel security exit under Windows

1 Write your own *usrsec.c* based on the samples delivered with EntireX.

2 Build your own *usrsec.dll*, using the provided makefiles.

Broker Stubs

To implement security exits for Broker stubs under Windows

1 Write your own *secuexit.c*, based on the samples delivered with EntireX.

2 Build your own *secuexit.dll*, using the provided makefiles.

Writing Security Exits

This section describes how to write your own security exits. It describes the interfaces, indicates what can be modified and what has to be done within an exit. It also provides some helpful tips.

This section covers the following topics:

Requirements

You must provide the following functions:

- The Preparation exit `etbupre()` and the Evaluation exit `etbueva()` for the Broker stub. These two functions are linked statically to the Broker stub routines.
- The Kernel exit `usrsec()` which is loaded by the kernel. This exit is more generic than the other two. It is called with the function that has been performed and a function-dependent Broker ACI control block that provides all the necessary information. This function is loaded dynamically by EntireX Broker during startup. One parameter of the kernel exit is the function that is performed.

The functions map to the exit type is as follows:

Exit Type	Function	Function to perform
Authentication exit	ETB_SEC_LOGON	Checks authentication for the user.
	ETB_SEC_LOGOFF	Release user-specific information if necessary.
	ETB_SEC_NEWPUID	Application call with different physical USER ID.
	ETB_SEC_NEWST	Application call with a different SECURITY TOKEN
Authorization exit	ETB_SEC_SEND	Check whether user is allowed to use the addressed service.
	ETB_SEC_REGISTER	Check whether the user is allowed to offer that service.
Encryption exit	ETB_SEC_ENCRYPT	Encrypt the given data. See Note 1 below.
	ETB_SEC_DECRYPT	Decrypt the given data. See Note 1 below.

Notes:

1. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.

In the following text, “encryption” or “authentication” exit refers to the functions listed above.

Error Checking for Incomplete Security Installation

With `ACI_VERSION=4` or above, the security configuration of the caller's stub is checked against the security configuration of the broker kernel. The request will be rejected with the error message `00200379 - API: Inconsistent Security Installation`, if security

- is present in the stub and it is not present in the kernel;

  or

- is not present in the stub and it is present in the kernel.

Note: If you have written your own security - instead of using *Security Solutions in EntireX* - and it is implemented only on the kernel, you will have to add a dummy security exit to the stub.

Security-Related Parameters

The following security-related parameters are provided. These are fields in the *Broker ACI Fields*:

- USER-ID
- PASSWORD
- SECURITY-TOKEN
- CLIENT-UID
- ERROR-CODE
- ERROR-TEXT
- KERNELSECURITY

USER-ID

The USER ID is defined by the application. It is available in all ACI exits as well as in the kernel exits, except the encryption and decryption exits. Theoretically the preparation exit can be used to retrieve the login name by an operating system specific mechanism. This would allow a user identification without the application being involved. See the description of the `USER-ID` field in the Broker ACI control block.

PASSWORD

The `PASSWORD` is defined by the application. It is available in all ACI and kernel exits except the encryption exit. The `PASSWORD`, if provided by the application in plain text, should be encrypted in the Preparation exit before sending it across insecure network connections. If the `PASSWORD` is needed in the application again, it must be decrypted after receipt in the Evaluation exit. The authentication exit must ensure that the `PASSWORD` is properly decrypted if necessary before sending it to an external security system.

The EntireX Broker provides minimal encryption of the `PASSWORD` field, that is, the field is not transmitted in plain text. If your environment requires absolute security, however, you will need to provide both Broker stub and EntireX Broker kernel exits to perform encryption and decryption. See the description of the `PASSWORD` field in the Broker ACI control block.

SECURITY-TOKEN

The SECURITY TOKEN can be created by the application and sent to EntireX Broker. That allows for a kind of credential algorithm. The security token is passed to all kernel exits and can therefore contain security information which is also important for the authorization and encryption exits. The SECURITY TOKEN can be altered in the authentication exit to provide a context token for that application and that session. It is transmitted back to the application and can then be used in all subsequent calls. For each subsequent call, the EntireX Broker checks whether the SECURITY TOKEN is identical to the one returned from the last call to the authentication exit, which could be the ETB_SEC_LOGON, the ETB_SEC_NEWPUI or the ETB_SEC_NEWST function. After an ETB_SEC_LOGOFF call, a subsequent call is always a ETB_SEC_LOGON call. See the description of the SECURITY-TOKEN field in the Broker ACI control block.

CLIENT-UID

CLIENT-UID is returned to a server application after a RECEIVE and contains the user ID of the sending client. This allows for further security checks (logging, separate checks, etc.). See the description of the CLIENT-UID field in the Broker ACI control block.

ERROR-CODE

All security-related ERROR CODEs start with the ERROR CLASS 0008. The following four digits in the ERROR CODE can be assigned by any exit if a security violation occurs. These digits only reach the application if the current operation is aborted by the security exit with a security violation, i.e. an appropriate return code. See ERROR-CODE under *Broker ACI Fields*.

ERROR-TEXT

The security exits can also pass an error message back to the application. This error text must not be longer than 40 bytes.

KERNELSECURITY

See KERNELSECURITY under *Broker ACI Fields*.

Programming Broker Stub Exits

The exits in the stub have the following interface:

- Preparation Exit
- Evaluation Exit
- Programming the Kernel Exit Routine

Preparation Exit

Synopsis

```
int etbupre (ETBCB *pEtbCb,
    void *pSendBuf,
    void *pReserved,
    char *pErrText)
```

Parameter	Format	Direction	Description
Address of ETBCB	Pointer to ETBCB control block.	I/O	ETBCB's user_id and password are used to generate the credential which will be saved in field security_token for function LOGON.
Address of send buffer	void pointer	I/O	Send buffer (3) supplied by caller, only available for function SEND, length of send buffer is member of ETBCB.
Reserved	void pointer	I	Must be NULL.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text that will be returned by the stub routine.

Return value

0 (okay) or non-zero (error)

The real error code must be written to the Broker control block as an 8-byte character array (without trailing 0 byte!). The Message Class 0008 - EntireX ACI - Security Error is reserved for all errors in function `etbupre`. The error number is user-defined. Additionally, the error text can be returned to the user in the error text array.

Required Actions in the Exit

If no data encryption is desired, no action is required.

Possible Actions in the Exit

- Generate a credential if function is `LOGON` and move it to the field security_token.
- Encrypt the send buffer if function is `SEND`. The encryption process must not change the length of the buffer. See Notes below.

The exit gets control for each function of ACI version 2 and above. The exit must exist.

Notes:

1. We recommend *not* implementing your own encryption/decryption mechanism. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.
2. The mainframe stubs (e.g. z/OS, BS2000) employ high performance direct cross memory to send and receive data from buffers in the application's working storage. This is utilized for sending/receiving more than 32 KB of data. Therefore, when encryption is active, the application programmer must not rely on the contents of the `SEND` buffer after issuing the `SEND` command, since the contents of the `SEND` buffer will be encrypted when sending more than 32 KB of data. We recommend you code all applications so that you do not rely on the contents of the `SEND` buffer after calling Broker. This will be required in the future for all `SEND` commands regardless of whether the data exceeds 32 KB. Therefore, the application's `SEND` buffer must not be in read-only memory, where encryption is activated.
3. A single send/receive buffer is used to perform encryption in place. This means that encrypted data is provided in the send buffer.

Evaluation Exit

Synopsis

```
int etbueva (ETBCB *pEtbCb,
    void *pRecBufEncr,
    void *pReserved,
    char *pErrText)
```

Parameters

Parameter	Format	Direction	Description
Address of ETBCB	Pointer to ETBCB control block.	I/O	ETBCB's security token is used for data decryption.
Address of receive buffer	void pointer	I/O	Receive buffer (2) provided by EntireX Broker. Only available for functions `RECEIVE` and `SEND WAIT=x` (implicit receive). Length of receive buffer is member of ETBCB.
Reserved	void pointer	I	Must be NULL.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text which will be returned by the stub routine.

Return Value

0 (okay) or non-zero (error)

The real error code must be written to the Broker control block as an 8-byte character string (without trailing 0 bytes!). The Message Class 0008 - EntireX ACI - Security Error is reserved for all errors in function `etbueva`. The error number is user-defined.

In addition, the error text can be returned to the user.

Required Actions in the Exit

If no data decryption is wanted, no action is required.

Possible Actions in the Exit

- Decrypt (1) the receive buffer if functions are `RECEIVE` or `SEND` with `WAIT`. The decryption process must not change the length of the buffer.

The exit gets control for each function of ACI Version 2 and above. The exit must exist.

Notes:

1. We recommend *not* implementing your own encryption/decryption mechanism. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.
2. A single send/receive buffer is used to perform decryption in place. This means that decrypted data is provided in the receive buffer. After the completion of a send/nowait command, the application should ignore the contents of the receive buffer.

Programming the Kernel Exit Routine

All authentication, authorization, encryption [(1)] and decryption exits are combined within one exit module named `USRSEC`. The various security checks are indicated by a type parameter in the CALL interface. `USRSEC` is provided with EntireX Broker as the default security exit. It is invoked if `SECURITY=YES` is set in the attribute file. Prior to EntireX, the `USRSEC` exit was available only with the SAF Gateway security package.

The general syntax of this user exit is defined as follows:

Synopsis

```
long usrsec (ETB_SECPAR *pParSec,
    void  *pVarious,
    char  *pErrText,
    char  *pWorkArea,
    long  lWorkArea)
```

Parameters

Parameter	Format	Direction	Description
Address of security parameter block	Pointer to structure ETB_SECPAR	I	Contains the security type flag.
Address of type-dependent security parameter block	void pointer	I	See control block structures `ETB_SECPAR_<type>`.
Address of error text	char pointer	O	The error text is an array of 40 characters containing the error text which will be returned to the user.
Address of work area	char pointer	O	Volatile work area.
Length of work area	long integer value	I	Size of the work area in number of bytes.

Return Value

0 (okay) or user-defined error number

Message Class 0008 - EntireX ACI - Security Error and the error number will be returned to the user. In addition, the error text can be returned to the user.

Notes:

1. We recommend *not* implementing your own encryption/decryption mechanism. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.

Required/Recommended Actions in the Exit (depending on Security Type)

Security Type	Required Action	Recommended Action	Notes
ETB_SEC_ENCRYPT	Copy decrypted to encrypted buffer and set the length of encrypted buffer. This is necessary because exit is called whether the receive data has to be encrypted or not.	Encrypt receive data if needed.	1,2
ETB_SEC_DECRYPT	Copy encrypted to decrypted buffer and set the length of decrypted buffer. This is necessary because exit is called irrespective of whether send data is encrypted or not.	Decrypt receive data if needed.	1,2
ETB_SEC_LOGON		Decrypt the password and check combination of user ID and password against the security system. Generate a context token according to the credentials of the user and EntireX Broker. Create a security handle for a user session (e.g. ACEE on z/OS).	
ETB_SEC_LOGOFF	None	Delete the security handle of the user session.	
ETB_SEC_NEWPUID	None	An application has changed the physical user ID between two calls. If necessary, a new security token can be created.	
ETB_SEC_NEWST	None	For some reason, the security token of an application has changed and no longer matches the original. The security token should be recalculated and approved or the application should be rejected.	
ETB_SEC_REGISTER	None	Check whether user_id is authorized to offer the requested SERVICE (check security_token data if necessary).	
ETB_SEC_SEND	None	Check whether user_id is authorized to offer the requested SERVICE (check security_token data if necessary).	
ETB_SEC_SECURE	None	Check the security token.	
ETB_SEC_INIT	None	Create global context if required.	
ETB_SEC_FINI	None	Delete global context if created.	
ETB_SEC_INFO	None	Check CIS information request.	
ETB_SEC_CMD	None	Check CIS command request.	

Notes:

1. The size of the buffer cannot be changed in this exit.
2. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS and Certificates with EntireX*.

V

10 EntireX Broker Tutorial

EntireX Broker is delivered with a Natural tutorial. This tutorial is written in the programming language Natural but is useful even if you are using another programming language. Natural is required for installation of the tutorial.

See also ***Examples for EntireX Broker Tutorial***.

Introduction to the Tutorial

The Natural tutorial shows you how to actively use EntireX Broker by

- allowing you to specify values for the fields in the ACI, which allows you to issue all types of requests and test use of EntireX Broker. See ***ACI Test Tool: Single Broker Request***.
- allowing you to measure throughput and response time of EntireX Broker. See ***Stress Mode***.
- offering several example client and appropriate server programs for programming language Natural; see ***Examples for EntireX Broker Tutorial***. All programs can be displayed, edited and executed. Help texts are available for each program to explain the purpose of the program, indicate typical usage, and illustrate the logical program flow.

Under UNIX and Windows, use the Natural SYSOBJH utility to install the EntireX Broker Tutorial (the Natural-based tutorial application `SYSETB` that is provided with EntireX). See *Object Handler* in the Natural Tools and Utilities documentation for more information.

Calling the Tutorial Menu

To activate the online tutorial, log on to library SYSETB in your Natural environment and issue the MENU command. This displays the online tutorial menu, which consists of a list of the client and server example programs:

```
18:54:34                  *** ENTIREX BROKER TUTORIAL ***           07-11-15

    Client   Server
   --------  --------   NON CONVERSATIONAL EXAMPLES -----------------------
__ EXCL01CP  EXCL01SP   Single Requests without Reply
__ EXCL03CP  EXCL03SP   Single Requests with Reply
   --------  --------   Conversational Examples ---------------------------
__ EXCN01CP  EXCN01SP   Long running Service - Non-blocked Client
__ EXCN02CP  EXCN02SP   Transfer messages from Server to Client
__ EXCN04CP  EXCN04SP   Transfer messages from Client to Server
__ EXCN05CP  EXCN05SP   Server with multiple parallel Conversations
   --------  --------   Special Features ----------------------------------
__ EXDM01CP  EXDM01SP   Send messages with HOLD - delayed delivery
__ EXDM02CP  EXDM02SP   Remove Service while Conversations exist
__ EXDM03CP  EXDM03SP   Server for multiple Services
   --------  --------   Customized Client/Server computing ----------------
__ EXRQ01-P  EXRQ01-P   Single Broker Requests
__ NATEX1CP  NATEX1SP   Model to write Client/Server programs API Version 1
__ NATEX2CP  NATEX2SP   Model to write Client/Server programs API Version>1

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  GLOB  EXIT                   UP    DOWN
```

The example programs are grouped according to the following types:

- *Non-conversational Examples*
- *Conversational Examples*
- *Special Features*
- *Getting Started*

Meaning of the information in the columns:

Column	Source
Client	Name of the client program
Server	Name of the server program followed by a description of the example.

Function keys available from the main menu:

PF Key	Function	Description
PF9	HELP	A general help is displayed.
PF2	GLOB	Prompts for global defaults to be used for the current session.
PF3	EXIT	Leave the online tutorial.
PF7	UP	Scroll up.
PF8	DOWN	Scroll down.

Global Defaults for the Tutorial

The following pop-up window is displayed when you press PF2 from the tutorial main menu:

```
   18:54:38              *** ENTIREX BROKER TUTORIAL ***              07-11-15

   Clie    +-------------------------------------------------------------------+
  -----    !  Please modify defaults or press ENTER to continue ...            !
__ EXCL0    !                                                                   !
__ EXCL0    !  Broker ID ......  ETBxxx                                         !
  ----      !  Server Class ...  ETB                                            !
__ EXCN0    !  Server Name ....  Tutorial                                       !
__ EXCN0    !  Broker Stub ....  BROKER                                         !
__ EXCN0    !  User ID ........  USR1                                           !
__ EXCN0    !  Token ..........                                                 !
__ EXCN0    !                                                                   !
  -----     !  Node ...........  Node: MVS/ESA     Name put into send data     !
__ EXDM0    !  Msg Length .....  64               Length of send/receive data !
__ EXDM0    !                                                                   !
__ EXDM0    !  Wait Time ......  45S              Time blocked SEND/RECEIVE   !
__ EXDM0    !  UOW Status Life.                   Life time 'addifier'        !
__ EXDM0    !  API Version       8                1, 2, 3, 4, 5, 6, 7, 8, 9   !
__ EXDM0    !                                                                   !
  -----     !  Locale String...                                                 !
__ EXRQ0    !                                                                   !
__ NATEX    !  Force Logon ....                   'Y', 'N'                    !
__ NATEX    !  Compress Level..                   '0' thru '9', 'Y', 'N'      !
__ NATEX    !  Kernel Security.                   'Y', 'N', 'U'               !
            +-------------------------------------------------------------------+

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  GLOB  EXIT                   UP    DOWN
```

The following global default settings can be modified and will be valid for the current session:

Global Default Setting	Meaning
Broker ID	ID of the Broker in use.
Server Class	Server class in use for every example.
Server Name	Server name in use for every example.
User ID	User ID in use when running an example.
Token	Token in use when running an example.
Node	Node name put into send data.
Msg Length	Message length used for the `SEND-LENGTH` and `RECEIVE-LENGTH`.
Wait Time	Timeout value used for blocked `SEND` and `RECEIVE` calls.
UOW Status Life	UOW Status lifetime. See *Writing Applications: Units of Work*.

Global Default Setting	Meaning
API Version	API version of broker control block. See *API-TYPE and API-VERSION*.
Locale String	Locale string. To be used to override or provide codepages. See *Using Internationalization*.
Force Logon	Override Broker AUTOLOGON. See *Authentication*.
Compress Level	Compression level. See *Data Compression*.
Kernel Security	Kernel security. See *Writing Applications using EntireX Security*.

Tutorial Commands

From the tutorial menu you can execute, list and edit example programs. You can also display several help texts on each program.

You can perform a function by entering the appropriate line command in the input field preceding the client program name. To display a list of available line commands, enter an asterisk in the input field preceding the client program name.

The table below lists the available line commands:

Command	Meaning
XC	Execute client program.
XS	Execute server program.
SH	Shut down server.
H	Help for the example as a whole.
HC	Help for client program.
HS	Help for server program.
LC	List (display) client program.
LS	List (display) server program.
EC	Edit client program.
ES	Edit server program.

The examples are also documented in *Examples for EntireX Broker Tutorial*.

Using the Tutorial Help

The tutorial help facility provides help texts for each client and server example program. To display the online help text, issue the appropriate line command, H, HC or HS, for the selected example on the online tutorial menu.

The following screen shows the online help for the server of the example "Single Requests without Reply" (line command HS):

```
19:08:25               *** ENTIREX BROKER Tutorial ***               03-05-15
                     Server: Single Requests without Reply

Descr. : This server establishes a service which is able to collect
         simple messages from clients that require no reply.
         A REGISTER is necessary to inform the Broker of the availability
         of the service. The DEREGISTER, issued as the last action, informs
         the Broker of the unavailability of the service served by this
         server.
         The server wants to wait for a client message and therefore uses
         a blocked RECEIVE, that is, a RECEIVE with W=nS is issued to the
         Broker.

Coding : LOGON    ------> logon to Broker
         REGISTER ------> offer service
         repeat
            RECEIVE,W=nS,CID=NEW ------> wait for message
         until ...
         DEREGISTER ------> deregister service
         LOGOFF ------> logoff from Broker

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP        EXIT                                Expml  Cln  Srv
```

The following functions are available from the help screen. You can execute a function by pressing the appropriate PF key:

PF Key	Function	Description
PF1	HELP	Display general help.
PF3	EXIT	Leave the help screen.
PF9	EXMPL	Display general help screen specific to example.
PF10	CLN	Display client help screen specific to example.
PF11	SRV	Display server help screen specific to example.

Note: You can use PF10 and PF11 to toggle between the client and server help screens.

Using the Example Programs

Use of the example “client/server programs” is the same for each example. You need to start two sessions in order to “play” with EntireX Broker, one by executing the server program and the other by executing the client program.

As the first session, start the server by entering `XS` in the input field preceding the program name, for example in the line for Single Requests without Reply. This displays the following startup parameter pop-up window:

```
9:11:38             *** ENTIREX BROKER TUTORIAL ***                 07-11-15
                              VERSION 8.0
    Client   Server
   -------- --------  NON CONVERSATIONAL EXAMPLES -----------------------
xs EXCL01CP  +-------------------------------------------------------+
__ EXCL03CP  !  Please enter values or press ENTER to continue ...   !
   --------  !                                                       ! ----
__ EXCN01CP  !  Mode ......... 1 1=Step 2=Stress 3=Silent           !
__ EXCN02CP  !                                                       !
__ EXCN04CP  !  Server Class . ETB                                   !
__ EXCN05CP  !  Server Name .. Tutorial                            !
   --------  !  Service ...... NcNoReply                           ! ----
__ EXDM01CP  !                                                       !
__ EXDM02CP  !  User ID ...... ILGWBU                              !
__ EXDM03CP  !  Token .........                                     !
   --------  !                                                       ! ----
__ EXRQ01-P  !  Msg Length ... 64                                   !
__ NATEX1CP  !                                                       ! n 1
__ NATEX2CP  +-------------------------------------------------------+ n>1

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  GLOB  EXIT                    UP   DOWN
```

The fields in this window are listed in logical groups. The first group controls the execution of the example and contains the mode parameter; for clients, end criteria to stop the execution is also displayed. Valid mode parameters are ***Step Mode***, ***Stress Mode***, ***Silent Mode***. The other fields show some global defaults which you can overwrite for this particular client/server run. Note, however, that the Broker ID and Wait Time values can only be modified in the Global Defaults window (see above).

When using an example for the first time, you are recommended to select Step mode.

Step Mode

In this mode, the example is executed step by step. This means that every broker call is displayed on your screen and must be explicitly issued by pressing PF5. Upon return, the response from the broker is displayed in the Errtxt field together with the next meaningful broker call, ready for execution. You can always view previous Broker calls using the trace facility (PF4), which provides "before and after" images of every call issued to the broker.

If you select Step Mode and press ENTER, a screen similar to the following is displayed for every example:

```
Press PF5 to issue Request ...
19:13:53                    *** ENTIREX BROKER TUTORIAL ***          03-05-15
                           Server: Single Requests without Reply

Errtxt .....
Send Data .. ____________________________________________________________________
Rcve Data ..

Type/Vers .. 1 / 5
Broker ID .. ETBxxx             Send Len ..... 64
Function ..* LOGON___________   Rcve Len ..... 64
Option ....* ________________   Errtx Len .... 40
Wait ......* _________          Rtrn Len ..... 0

Class ...... ETB_____________   User ID ...... ILGWBU_____________________________
Name ....... Tutorial________   Token ........ ____________________________________
Service .... NcNoReply_______   Password ..... ____________________________________
Conv ID ...* ________________   New Password . ____________________________________
User Data .. ________________   Sec Token .... ____________________________________
Conv Stat .. Environment ..     ___________________________________
                                Client UID ...

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help Strss  Exit  Trace Exec                          SBuff RBuff
```

The following functions are available from this screen. You can execute a function by pressing the appropriate PF key:

PF Key	Function	Description
PF1	HELP	Display the help screen on the example program. See *Using the Tutorial Help*.
PF2	STRSS	Change execution mode to Stress.
PF3	EXIT	Leave sample program.
PF4	TRACE	Invoke the *The Tutorial Trace Facility*.
PF5	EXEC	Issue broker call.
PF10	SBUFF	See *Display/Modify Send Buffer*.
PF11	RBUFF	See *Display/Reset Receive Buffer*.

Stress Mode

In this mode, the example is executed without further user interaction. Every Broker call issued is also displayed on the screen to allow you to see the activity of the client or server. Execution terminates in different ways:

- **For clients:**
  Further end criteria (such as number of messages and number of conversations) are supplied in the startup parameter window of the client example. When the specified values are reached, processing stops.
- **For servers:**
  Servers run until they are shut down by a special shut down message sent to the server (`SH` command from the tutorial main menu).

When execution in Stress mode is stopped, the following summary of client/server activity is displayed:

```
Waiting for Request ...
20:54:37                 *** ENTIREX BROKER TUTORIAL ***               03-05-15
                       Server: Single Requests without Reply
+-----------------------------------------------------------------------------+
!                                                                             !
!   00200216 API: Invalid BROKER-ID                                           !
!   OP System .. MVS                          Load Count Max                  !
!   TP System .. CICS                         ------------ -------- -------- !
!   Speed/Mode . 191.850 / 2                  Messages ...                    !
!   Msg Length . 64                           Conv .......                    !
!   ETB Calls .. 1                            Parallel CID                    !
!                                                                             !
!   Time/Call     Count  Ave     Min     Max    Time elapsed Absolute Relative !
!   ------------ ------ ------- ------- ------- ------------ -------- -------- !
!   Send non-blk                              Total ...... 0.0       100  %  !
!   Send blocked                              Executing .. 0.0       83.5 %  !
!   Rcve non-blk                              Waiting                       !
!   Rcve blocked                              Transport . 0.0        16.4 %  !
!   EOC ........                              Partner ... 0.0             %  !
!   Undo .......                                                            !
!   Register ...                                                            !
!   Deregister .                                                            !
+-----------------------------------------------------------------------------+
   Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
         Help  Strss Exit Trace Exec                          Buff RBuff
```

Meaning of the fields:

Field	Meaning
OP	System Underlying operating system.
TP	System Underlying transaction monitor.
Speed	Indication of the performance of the environment, relative to the corresponding value of other environments.
Mode	Execution mode of the example.
Msg	Length of messages sent/received.
ETB	Number of calls issued to the broker.
Load	
Messages/Count	Number of messages sent/received.
Messages/Max	Number of messages used as criteria to stop execution.
Conv/Count	Number of conversations conducted.
Conv/Max	Number of conversations used as criteria to stop execution.
Parallel CID/Count	Highest number of parallel conversations reached.
Parallel CID/Max	Maximum number of parallel conversations allowed.
Time/Call	
Send non-blk/Count	Number of non-blocked SEND calls issued.
Send non-blk/Ave	Average elapsed time for a non-blocked SEND call.
Send non-blk/Min	Shortest elapsed time for a non-blocked SEND call.
Send non-blk/Max	Longest elapsed time for a non-blocked SEND call.
Send blocked	Same as above for blocked SEND calls.
Rcve non-blk	Same as above for non-blocked RECEIVE calls.
Rcve blocked	Same as above for blocked RECEIVE calls.
EOC	Same as above for EOC calls.
Undo	Same as above for UNDO calls.
Register	Same as above for REGISTER calls.
Deregister	Same as above for DEREGISTER calls.
Time elapsed	
Total/Absolute	Elapsed time in seconds between start and end for the run.
Total/Relative	Percentage of time between start and end for the run.
Executing/Absolute	Elapsed time in seconds when example is executing.
Executing/Relative	Percentage of time when example is executing.
Waiting	Time needed for transport plus execution time required by the partner.
Transport/Absolute	Elapsed time in seconds used for transport services. Transport means EntireX Broker and all other media involved such as SVCs, link routines, Entire Net-work, TCP/IC.
Transport/Relative	Percentage of time used for transport services.

Field	Meaning
`Partner/Absolute`	Elapsed time in seconds needed by the partner to execute the call. This is relevant only to blocked `SEND` calls, as this is the only call involving a partner.
`Partner/Relative`	Percentage of time needed by the partner to execute the call. This is relevant only to blocked `SEND` calls, as this is the only call involving a partner.

Note:

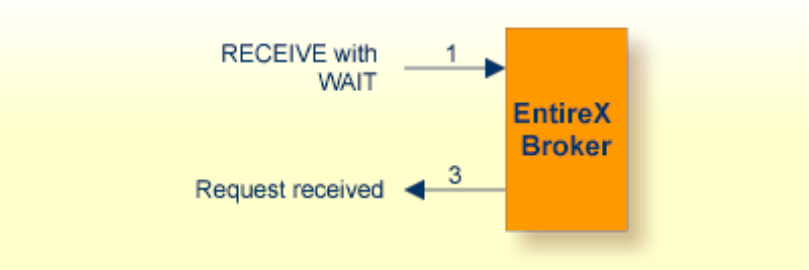
`Total/Relative=(Executing/Relative)+(Transport/Relative)+Partner/Relative=100%`

The waiting period of the different call types consists of the following times:

Blocked RECEIVE

For blocked `RECEIVE`s, the elapsed time is calculated from the following:

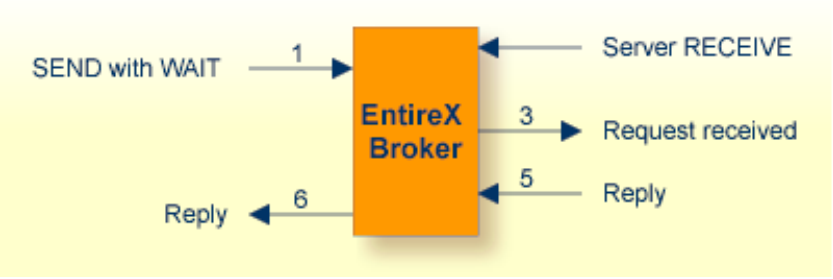
1. The time when the `RECEIVE` call was routed from the server to the broker.
2. A time of no activity during which there was no client request to be processed. This value may be high.
3. The time when an incoming client request was routed from the broker to the server.



Blocked SEND

For blocked `SEND`s, the elapsed time is calculated from the following:

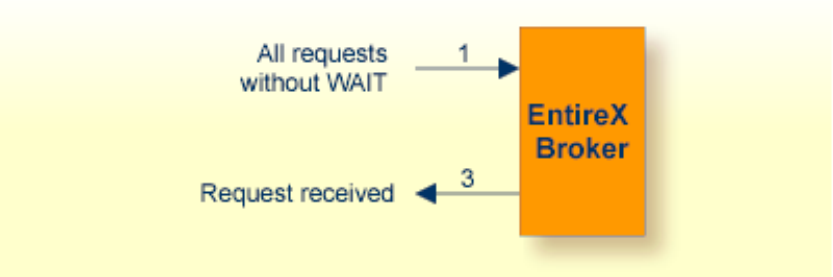
1. The time when the `SEND` call was routed from the client to the broker.
2. A time of no activity during which there was no server ready to process the request. This value may be high.
3. The time when the client request was routed from the broker to the server.
4. The time when the request was processed by the server.
5. The time when the response was routed from the server to the broker.
6. The time when the answer was routed from the broker back to the client.



All Other EntireX Broker Calls

For all other calls to the broker, the elapsed time is calculated from the following:

1. The time when the call was routed from the participant to the broker.
2. The time when the call was processed by the broker.
3. The time when the call was routed from the broker back to the participant.



Silent Mode

In this mode, the same applies as for Stress mode, except that no map I/Os are performed between broker calls. It is therefore not possible to view activities while the client and server example is running.

The Tutorial Trace Facility

The trace facility is activated by pressing the appropriate PF key after starting an example program. With the trace option on, “before and after” images of the last ten requests issued to the broker are made visible. When the trace option is selected, the most recent request is always displayed:

Use PF7 / PF8 to scroll to older / more recent requests. Scroll right with PF11 to display a second screen page for every request.

```
21:00:07              *** ENTIREX BROKER TUTORIAL ***                 03-05-15
------------ Image after call --------------- Image before call -    0 First
Type/Vers .. 1 / 5                            1 / 5
Errtext .... 00000000 Successful response
------------ -------------------------------- --------------------------------
Broker ID .. ETB233 ETB233
Class ...... ETB ETB
Name ....... Tutorial Tutorial
Service .... NcNoReply NcNoReply
Fct ........ LOGON LOGON
Option .....
Wait .......
Conv ID ....
Conv Status.
User Data ..
Client UID .
------------ -------------------------------- --------------------------------
Send Data .. 0000000000326891781
Rcve Data ..
------------ -------------------------------- --------------------------------

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                   Up    Down        Left  Right
```

The following functions are available from this screen You can execute a function by pressing the appropriate PF key:

PF Key	Function	Description
PF1	HELP	Display a help screen on the example program.
PF3	EXIT	Leave trace.
PF7	UP	Scroll to older requests.
PF8	DOWN	Scroll to more recent requests.
PF10	LEFT	Scroll to first screen page.
PF11	RIGHT	Scroll to second screen page.

Display/Modify Send Buffer

Selecting this option by pressing PF10 after starting the example from the tutorial menu displays the send buffer contents in hexadecimal and character format:

```
21:01:28               *** ENTIREX BROKER TUTORIAL ***              03-05-15
                          Display/Modify Send Buffer

00016 0000000000081804 F0F0F0F0F0F0F0F0F0F0F0F8F1F8F0F4    Send Len .. 64
00032 070_____________ F0F7F040404040404040404040404040
00048 _______________ 40404040404040404040404040404040
00064 _______________ 40404040404040404040404040404040
00080 _______________ 40404040404040404040404040404040
00096 _______________ 40404040404040404040404040404040
00112 _______________ 40404040404040404040404040404040
00128 _______________ 40404040404040404040404040404040
00144 _______________ 40404040404040404040404040404040
00160 _______________ 40404040404040404040404040404040
00176 _______________ 40404040404040404040404040404040
00192 _______________ 40404040404040404040404040404040
00208 _______________ 40404040404040404040404040404040
00224 _______________ 40404040404040404040404040404040
00240 _______________ 40404040404040404040404040404040
00256 _______________ 40404040404040404040404040404040
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit              Top   Up    Down  Bot    Posit      Reset
```

Use PF6 to PF9 to scroll up or down as needed. Positioning to a specific offset is possible by pressing PF10. You can overwrite the send buffer contents in the character-oriented column. The send buffer is cleared with PF12.

Meaning of the information in the columns from left to right:

Column	Meaning
1	Send buffer offset decimal.
2	Send buffer contents displayed in character format.
3	Send buffer contents displayed in hexadecimal format.

The following functions are available from this screen. You can execute a function by pressing the appropriate PF key:

PF Key	Function	Description
PF1	HELP	Display a help screen on the example program.
PF3	EXIT	Leave send buffer display.
PF6	TOP	Position to first page.
PF7	UP	Scroll one up page.
PF8	DOWN	Scroll down one page.
PF9	BOT	Position to last page.
PF10	POSIT	Position to a specified offset in the send buffer.
PF12	RESET	Set the send buffer to low values.

Display/Reset Receive Buffer

Selecting this option by pressing PF11 after starting the example from the tutorial menu displays the receive buffer contents in hexadecimal and character format in the same way as for the send buffer. See See *Display/Modify Send Buffer*.

ACI Test Tool: Single Broker Request

This screen is an ACI test tool. An interface is provided which allows you to fill the broker ACI yourself and therefore issue all types of ACI requests in any sequence. You can use it

- for test purposes of EntireX Broker;
- for studying EntireX Broker functions and functionality;
- as counterpart of any client or server written in any programming language.

If you execute this program, (line command XC or XS), the user interface presents the broker ACI directly, which you can fill:

```
Press PF5 to issue Request ...
19:46:24                    *** ENTIREX BROKER TUTORIAL ***          03-05-15
                                : Single Broker Requests

Errtxt .....
Send Data .. ____________________________________________________________________
Rcve Data ..

Type/Vers .. 1 / 5
Broker ID .. ETBxxx             Send Len ..... 0
Function ..* ________________   Rcve Len ..... 0
Option ....* ________________   Errtx Len .... 40
Wait ......* ________           Rtrn Len ..... 0

Class ...... ETB_____________   User ID ...... ILGWBU_____________________________
Name ....... Tutorial________   Token ........ ____________________________________
Service .... Request_________   Password ..... ____________________________________
Conv ID ...* ________________   New Password . ____________________________________
User Data .. ________________   Sec Token .... ____________________________________
Conv Stat ..                    Environment .. ____________________________________
Client UID ...

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit Trace  Exec  Reg   Dreg  Send  Rcve  SBuff RBuff Reset
```

Press PF6 to PF9 to assign default values to the broker ACI for the selected function. A field help is available for fields marked with an asterisk (mark the field with the cursor and press PF1).

To issue a request to the broker, press PF5.

The following functions are available from this screen. You can execute a function by pressing the appropriate PF key:

PF Key	Function	Description
PF1	HELP	Display a help screen on this example program. If you press PF1 with the cursor on a field marked with an asterisk (*), a help window for the field is displayed.
PF3	EXIT	Leave the program.
PF4	TRACE	Invoke tracing of requests. See *The Tutorial Trace Facility*.
PF5	EXEC	Route a request to the broker.
PF6	REG	Assign defaults for REGISTER function to the ACI.
PF7	DREG	Assign defaults for DEREGISTER function to the ACI.
PF8	SEND	Assign defaults for SEND function to the ACI.
PF9	RCVE	Assign defaults for RECEIVE function to the ACI.
PF10	SBUFF	See *Display/Modify Send Buffer*
PF11	RBUFF	See *Display/Reset Receive Buffer*:
PF12	RESET	Set the ACI to low values.

11 Examples for EntireX Broker Tutorial

This chapter documents the examples provided in the EntireX Broker Tutorial. The purpose of each example is outlined, the objective of the client and server parts of the example is explained, and the logical program flow is illustrated. This should help you implement similar functionality using any of the supported programming languages. The Online Tutorial contains Natural example code to demonstrate these examples.

Non-conversational Examples

Example 1: Single Request without Reply

This example shows a client sending simple messages that do not require a reply from a server, for example feeding statistical performance data into a network-wide performance monitor. Since no reply is expected, the client does not have to wait for an answer and therefore issues a non-blocked `SEND` call to the broker. The established communication is non-conversational.

Such a client could be used as a trigger for a net management server from all servers in the network.

Client

The client issues simple messages to a server without expecting a reply. Because no reply is required (the server will not return any response), the client issues a `SEND` without wait (`W=NO`). This type of call is called non-blocked, and control is returned to the caller immediately. The client specifies non-conversational communication using "`NONE`" in the `CONV-ID` field of the ACI control block.

Server

The server establishes a service which is able to collect simple messages from clients that do not require a reply. A `REGISTER` is necessary to inform the Broker of the availability of the service. The `DEREGISTER`, issued as the last action, informs the Broker of the unavailability of the service served by this server.

The server wants to wait for a client message and therefore uses a blocked `RECEIVE` - that is, a `RECEIVE` with `W=nS` is issued to the Broker.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=NO,CID=NONE ------> forward message to server
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for message
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Example 2: Single Request with Reply

This example shows a client sending requests that require a reply from a server, for example a database access. Traditional remote procedure calls (RPCs) are also referred in this way. Since a reply is expected, the client uses a blocked `SEND` to issue the request to the server and wait for the reply. This is the equivalent of an implicit receive. The established communication is non-conversational.

Client

The client issues requests and expects a reply from the server. Because a reply is required and no conversation is built, a blocked `SEND (W=nS)` must be used. If the wait time elapses before the reply is received, there is no chance (in non-conversational mode) of getting the reply. However, you can retrieve the reply later in conversational mode by issuing a subsequent `RECEIVE`.

Server

The server establishes a service that is able to receive requests and return a reply to the client. Although the communication is non-conversational, the server gets a conversation ID with the incoming request. This ID must be retrieved and used when sending back the reply to the client.

The server must issue the `RECEIVE` call with `CID=NEW` in order to prevent unnecessary "Conversation ID timeout" messages.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NONE ------> send and wait for reply
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for request
   SEND,W=NO,CID=1234 ------> reply to client
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Conversational Examples

Example 3: Long Running Service - Non-blocked Client

This example shows a client dealing with a long-running service. The server process is initiated with a non-blocked `SEND` request. Later on, the client checks the processing status with a non-blocked `RECEIVE` request. However, the client retains control in all broker calls and is never blocked. The established communication is conversational. This example applies to any background processing in which the client should retain control.

Client

The client issues a non-blocked `SEND` to initiate a conversation with the desired service. With the subsequent non-blocked `RECEIVE` requests, the process is checked to see if it is still running or has finished.

Server

The server provides a service which takes some time to finish. It demonstrates a non-blocked client example. The long running processing is simulated by a wait of 30 seconds done with a blocked `RECEIVE` to a dummy `WAIT` service.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=NO,CID=NEW ------> initiate process/conversation
   repeat
      RECEIVE,W=NO,CID=1234 ------> check for process status
      decide on ERROR-CLASS
         VALUE 0 successful response - retrieve reply
         VALUE 3 processing ended
         VALUE 74 wait some time and retry
   until ...
until ...
LOGOFF ----> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   wait 30 seconds - simulate long running processing
   SEND,OP=EOC,W=NO,CID=1234 -----> reply and EOC
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Example 4: Transfer Messages from Server to Client

This example shows a client retrieving a large amount of data from a server within a conversation, for example a `GET <file>` command of a file transfer system. The transfer of messages/data to the client is done by the server with non-blocked `SENDs`. This is important because the server can work independently from the client, that is, forward the data/messages to the client and is then quickly free to process the next conversation. The established communication is conversational.

Client

The client receives a large amount of data/messages from a server. The `SEND` initiates a conversation with the desired service. Following the `RECEIVE`, the client retrieves data/messages from the connected server until the conversation is ended by the server.

Server

The server is able to send a large amount of data/messages to the client. The data/messages are transferred with non-blocked (`W=NO`) `SENDs`. The last transfer terminates the conversation with a non-blocked `SEND` and option `EOC`.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NEW ------> initiate conversation
   repeat
      RECEIVE,W=nS,CID=1234 ------> receive data/message
      decide on ERROR-CLASS
         VALUE 0 successful response
         VALUE 3 conversation ended
   until ...
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   SEND,W=NO,CID=1234 ------> acknowledge conversation
   repeat
      SEND,W=NO,CID=1234 ------> transfer data/message
   until end of data
   SEND,OP=EOC,W=NO,CID=1234 -----> last data/message and EOC
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Example 5: Transfer Messages from Client to Server

This example shows a client transferring a large amount of data to a server using conversational communication, for example, a `PUT <file>` command of a file transfer system. Once the conversation is established, the server depends on the client's activity, because the client always sends the messages/data and finishes the conversation, thus tying the server to one conversation for a long time. This might in some circumstances be unacceptable. The situation can be improved when multiple servers for this service are started simultaneously.

Client

The client transfers a large amount of data/messages to the server. The first blocked `SEND` initiates a conversation with the server. The server acknowledges the conversation with a reply. Subsequent non-blocked SENDs then transfer the data/messages to the server. The last transfer terminates the conversation with a non-blocked `SEND` and option `EOC`.

Server

The server retrieves a large amount of data from the client. The server depends on the client at the second `RECEIVE` for the data/messages, because the call is blocked.

Coding

```
Client
LOGON -----> logon to Broker
repeat
   SEND,W=nS,CID=NEW ------> initiate conversation
   repeat
      SEND,W=NO,CID=1234 ------> transfer data/message
   until ...
   SEND,OP=EOC,W=NO,CID=1234 ------> last data/message and EOC
until ...
LOGOFF ----> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   SEND,W=NO,CID=1234 ------> acknowledge conversation
   repeat
      RECEIVE,W=nS,CID=1234 ------> receive data/message
   until ...
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Example 6: Server with Multiple Parallel Conversations

This example shows a server which is able to process multiple conversations in parallel. To build such a server, the states of active conversations must be maintained in order to know where processing continues when the next request/message for the conversation is retrieved. Be aware that this can lead to complicated programs (multiplexing servers in environments where it is not feasible to have one server process per client).

A simpler and more convenient way to build a server environment which is able to process multiple conversations is to start replicates of the server. However, multiplexing servers may be appropriate in environments with restricted resources (for example, limited number of tasks).

The established communication is conversational.

Client

This client is used to demonstrate a server which is able to process multiple conversations in parallel. The first blocked `SEND` initiates the conversation. The server always acknowledges the conversation with a reply to the client. With the subsequent calls, requests/replies are transferred within the established conversation. The conversation is terminated by issuing an EOC.

Server

The server processes multiple conversations in parallel. At the `RECEIVE` with `CID=ANY`, client requests are retrieved, which belong either to existing or new conversations. All known conversations are stored in an array. When conversations finish, these entries are freed. When the last entry is used, `CID=OLD` is issued, preventing the retrieval of new conversations.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NEW ------> initiate conversation
   repeat
      SEND,W=nS,CID=1234 ------> ongoing conversation
   until ...
   EOC,CID=1234 ------> end of conversation
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________________
Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=ANY/OLD ------> OLD if max parallel reached
   decide on ERROR-CLASS
      VALUE 0 successful response
```

```
         SEND,W=NO,CID=1234 ------> new or ongoing conversation
      VALUE 3 conversation ended
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Special Features

- Example 7: Send Messages with HOLD - Delayed Delivery
- Example 8: Remove Service while Conversations Exist
- Example 9: Server for Multiple Services

Example 7: Send Messages with HOLD - Delayed Delivery

This example demonstrates the HOLD facility of EntireX Broker. Data/messages are set in hold by the SEND with the option HOLD. This prevents the partner from retrieving the data/messages until a SEND without the HOLD option is issued. Held data/messages are always under control of the sender until they are released. With the function UNDO, the sender can remove held data/messages.

The HOLD option is useful if a packet of data has to be delivered that does not fit in one request. Either the whole request packet has to be shipped, or nothing (minimum transaction support). The established communication is conversational. To set data/messages in hold only makes sense in conversational communications.

Client

This client demonstrates the hold mechanism used by the server. The data/messages are set in hold by the server and released with the last data/message sent. The client does not recognize this.

Server

The server sends data using the HOLD facility. Data is set in hold with SEND and option HOLD.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NEW ------> initiate conversation
   repeat
      RECEIVE,W=nS,CID=1234 ------> receive data/messages
      decide on ERROR-CLASS
         VALUE 0 successful response
         VALUE 3 conversation ended
   until ...
```

```
until ...
LOGOFF ------> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   SEND,W=NO,CID=1234 ------> acknowledge conversation
   repeat
      SEND,OP=HOLD,W=NO,CID=1234 ------> set data in hold
   until ...
   if error
      UNDO ------> remove accumulated data
   endif
   SEND,OP=EOC,W=NO,CID=1234 -----> release data in hold
until ...
DEREGISTER ----> deregister service
LOGOFF ------> logoff from Broker
```

Example 8: Remove Service while Conversations Exist

This example demonstrates a server that deregisters while conversations still exist. The conversations continue. With the option `QUIESCE` used on the `DEREGISTER` function, servers are able to remove their services in a smooth way. Established conversations are allowed to continue until ended with `EOC` by any partner. This mechanism is needed to shut down a server without aborting existing conversations.

New conversations are not accepted for servers that have removed their services and will be connected by the broker to other servers if available, or else rejected.

The established communication is conversational.

Client

The client establishes a conversation with a blocked `SEND`. After retrieving an acknowledgment from the server, subsequent requests/replies are transferred within this conversation. However, the service is deregistered while the conversation continues.

Server

After a new conversation is retrieved, the server removes the service in a smooth way by issuing a DEREGISTER with option QUIESCE. The established conversation continues until ended by the client.

Coding

```
Client
LOGON ------> logon to Broker
SEND,W=nS,CID=NEW ------> initiate conversation
repeat
   SEND,W=nS,CID=1234 ---> ongoing after deregistration
until ...
EOC,CID=1234 -----------> end of conversation
LOGOFF -----> logoff from Broker
_____________________________________________________________
Server
LOGON ------> logon to Broker
repeat
   REGISTER ------> offer service
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   DEREGISTER,OP=QUIESCE ------> deregister service
   SEND,W=NO,CID=1234 ------> acknowledge new conversation
   repeat
      RECEIVE,W=nS,CID=1234 ------> ongoing conversation
      SEND,W=NO,CID=1234 ------> reply to client
   until 3 conversation ended
until ...
LOGOFF ------> logoff from Broker
```

Example 9: Server for Multiple Services

This example demonstrates a server offering multiple services. It is possible to issue a RECEIVE to the broker and specify the service name with an asterisk(*) in any of the fields SERVER-CLASS, SERVER-NAME and SERVICE. This enables clients to wait for multiple services with one RECEIVE. The services waited for must all be previously registered. The asterisk(*) notation can also be used in DEREGISTER calls.

This feature is useful for alias service names or multipurpose servers. For example, a server might be able to retrieve data from a database, to add data and to remove data. A way to implement this is to register three different services.

The established communication is non-conversational.

Client

This client demonstrates a server which is able to offer multiple services. The name of the service the message is routed to is alternately switched between Service 1 and Service 2.

Server

The server demonstrates how to offer multiple services. With the `REGISTER` call, two services are established. With the `RECEIVE` call using the asterisk notation for the service (`SV=*`), the server can process any request for any of the services it has registered. The actual service name to which the request belongs is returned in the `SERVER-CLASS`, `SERVER-NAME` and `SERVICE` fields by the Broker. This allows the server to offer multiple services with a single `RECEIVE` call.

With the `DEREGISTER` call, all previously registered services are removed using the asterisk notation for the service name (`SV=*`).

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,SV=SV1,W=nS,CID=NONE ------> send to first service
   SEND,SV=SV2,W=nS,CID=NONE ------> send to second service
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________

Server
LOGON ------> logon to Broker
REGISTER,SV=SV1 ------> offer first service
REGISTER,SV=SV2 ------> offer second service
repeat
   RECEIVE,SV=*,W=nS,CID=NEW ------> wait for any service
   SEND,W=NO,CID=1234 ------> reply to client
until ...
DEREGISTER,SV=* ------> deregister all services
LOGOFF -----> logoff from Broker
```

Getting Started

Example 10: ACI Test Tool: Single Broker Requests

This screen is an ACI test tool. An interface is provided which allows you to fill the broker ACI yourself and therefore issue all types of ACI requests in any sequence. You can use it

- for test purposes of EntireX Broker;
- for studying EntireX Broker functions and functionality;
- as counterpart of any client or server written in any programming language.

Example 11: Model to write Client/Server Programs API Version 1

This example shows a simple client/server communication. It implements *Single requests with Reply* (see also this example in the tutorial). The client issues a simple request and waits for a reply from the server.

The established communication is non-conversational.

The programs for this example do not need any other Natural object (maps, data areas etc.) for execution.

You can copy the programs to any Natural library and use them as models to write your own client/server programs.

Client

This client issues requests and expects a reply from the server. Because a reply is required and no conversation is built, a blocked `SEND (W=nS)` must be used (see also the example *Single Requests with Reply* in the tutorial).

You can copy this program to any Natural library and use it as model to write your own client programs.

Server

This server establishes a service which is able to collect simple messages from clients that do not require a reply. Although the communication is non-conversational the server gets a conversation ID with the incoming request. This ID must be used when sending back the reply to the client (see also the example *Single Requests with Reply* in the tutorial). You can copy this program to any Natural library and use it as a model to write your own server programs.

Coding

```
Client
repeat
   SEND,W=nS,CID=NONE ------> send and wait for reply
until ...
_____________________________________________________

Server
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for request
   SEND,W=NO,CID=1234 ------> reply to client
until ...
DEREGISTER ------> deregister service
```

Example 12: Model to write Client/Server programs API Version 2

This example shows a simple client/server communication. It implements *Single requests with Reply* (see also this example in the tutorial). The client issues a simple request and waits for a reply from the server.

The established communication is non-conversational.

The programs for this example do not need any further Natural object (maps, data areas etc.) for execution.

You can copy the programs to any Natural library and use them as models to write your own client/server programs.

Client

This client issues requests and expects a reply from the server. Because a reply is required and no conversation is built, a blocked `SEND (W=nS)` must be used (see also the example Single Requests with Reply in the tutorial).

You can copy this program to any Natural library and use it as model to write your own client programs.

Server

This server establishes a service which is able to collect simple messages from clients that do not require a reply. Although the communication is non-conversational the server gets a conversation ID with the incoming request. This ID must be used when sending back the reply to the client (see also the example *Single Requests with Reply* in the tutorial). You can copy this program to any Natural library and use it as a model to write your own server programs.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NONE ------> send and wait for reply
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________

Server
LOGON ---------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=NEW ------> wait for request
   SEND,W=NO,CID=1234 --------> reply to client
until ...
DEREGISTER ----> deregister service
LOGOFF --------> logoff from Broker
```

Attach Manager Interface

Example 13: Demonstration of the Attach Manager Interface

An Attach Manager is a server that is able to start server. If no server is found for a client request, the Broker informs the Attach Manager to start the desired server. To be informed by the Broker, the Attach Manager must previously register all servers for which it is responsible using the option `ATTACH`.

Coding

```
LOGON ------> logon to Broker
REGISTER ----------------------> Attach Manager main service
REGISTER,OP=ATTACH,SV=SV1 ------> attachable service
repeat
   RECEIVE,W=nS,CID=NEW ---------> wait for any service
until ...
DEREGISTER,SV=* ------> deregister all services
LOGOFF -----> logoff from Broker
```

Non-blocked Server

- Example 14: Single Requests without Reply - A Polling Server
- Example 15: Single Requests with Reply - A Polling Server

Example 14: Single Requests without Reply - A Polling Server

Demonstration of Attach Manager Interface:

This example shows a server collecting simple messages from clients that do not require a reply. The server polls for a message at the `RECEIVE`, i.e. the `RECEIVE` is not blocked. This enables the server to do other work, even if no message is available for processing. The client uses a non-blocked `SEND` because no reply is expected from the server. The communication is non-conversational.

Example

A Server collecting cyclic statistical data from various input media, e.g. mainframe console, job management systems, databases and client messages from the broker.

Client

This client issues simple messages to a server without expecting a reply. Because no reply is required - the server will not return any response - the client issues a `SEND` without wait (`W=NO`). This type of call is called non-blocked because it is not blocked and control is returned immediately to the caller. With a value of "NONE" in the `CONV-ID` field of the ACI control block the client specifies non-conversational communication.

Server

This example shows a server collecting simple messages from clients that do not require a reply. The server polls for a message at the `RECEIVE`, i.e. the `RECEIVE` is not blocked. This enables the server to do other work, even if no message is available for processing. The client uses a non-blocked `SEND` since no reply is expected from the server. The communication is non-conversational.

A Server collecting cyclic statistical data from various input media, e.g. mainframe console, job management systems, databases and client messages from the broker.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=NO,CID=NONE ------> forward message to server
until ...
LOGOFF ----> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
RECEIVE,W=NO,CID=NEW ------> poll for message
decide on ERROR-CLASS
   VALUE 0 successful response
   VALUE 74 no message available - so free for other work
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Example 15: Single Requests with Reply - A Polling Server

This example shows a client sending requests/messages and expecting a reply from the server. The established communication is non-conversational. Because a reply is expected, the client uses a blocked `SEND` call to the broker. The server polls for a request at the `RECEIVE`, i.e. the `RECEIVE` is non-blocked. This enables the server to do other work, even if no request is available for processing.

Client

This client issues requests/messages and expects a reply from the server. Because a reply is required and no conversation is built, a blocked `SEND (W=nS)` must be used. If the wait time elapses before the reply is received, there is no chance in non-conversational mode of getting the reply. However, you can do this in conversational mode by issuing a subsequent `RECEIVE`.

Server

This server establishes a service that is able to receive requests/messages and return a reply to the client. The server works non-blocked at the `RECEIVE`, that is, a `RECEIVE` with `W=NO` is issued to the Broker. Because of this non-blocked call, control is retained, allowing the server to do other work.

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NONE ------> send and wait for reply
until ...
LOGOFF -----> logoff from Broker
_____________________________________________________________

Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=NO,CID=NEW ------> poll for request/message
   decide on ERROR-CLASS
      VALUE 0 successful response
         SEND,W=NO,CID=1234 ------> reply to client
      VALUE 74 no message available - so free for other work
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

VI

12 EntireX Broker Reporting

This chapter details the reporting options of EntireX Broker.

Configuration Report

EntireX Broker reads configuration information from an attribute file during startup. In order to reduce the number of different attribute files, you may define a global attribute file and specify the individual settings within a variable definitions file. Thus unique attributes like `BROKER-ID` and `PORT` are kept as part of the variable definitions file, while other parameters such as service definitions can be shared among a group of Broker instances. This feature is described in detail in *Variable Definition File*.

In the past there was a one-to-one relationship between Brokers and attribute files. To determine your Broker configuration, you could reference your attribute file. Now that you may create a global attribute file and substitute parameters at startup, it may not be clear what configuration was used to start your Broker. To see the exact configuration used at startup, you can now view the configuration report for each Broker. The configuration report will display exactly which values were used for each attribute at startup.

Here is a sample configuration report:

```
EntireX 8.0.0.12     Configuration Report      2007-10-02 08:56:23    Page     1

Variable definitions file:
   1: BID = ETB191
   2: N   = 113
   3: P   = HOT
   4: PCA = localhost:3938:SSL
   5: PT  = ADABAS
   6: RM  = STANDARD
   7: SP  = 3939
   8: TP  = 3930
   9: TR  = SSL-TCP-NET
EntireX 8.0.0.0     Configuration Report      2007-10-02 08:56:23    Page     2

Attribute file:
   1: ************************************************************************
   2: *                                                                      *
   3: *                    EntireX Broker Attribute File                     *
   4: *                                                                      *
   5: ************************************************************************
   6:
   7: ******************** Global section ***********************************
   8:
   9: DEFAULTS = BROKER
  10:   ABEND-MEMORY-DUMP              = NO
  11:   ACCOUNTING                     = NO
  12:   AUTOLOGON                      = YES
```

```
  13:   BROKER-ID                       = ${BID}
- - - - Substitution: ${BID} = ETB191
  14:   CLIENT-NONACT                   = 15M
```

The contents of the variable definitions file and the contents of the attribute file are copied to this configuration report. In addition, all variables in the attribute file will be appended by another line reporting the effective value for the variable. Thus, it's possible to keep track of the substitution procedure.

On UNIX and Windows, a file called `CONFIG.REPORT` is created in the current working directory of Broker. The environment variable `ETB_CONFIG_REPORT` may contain an alternative location. However, on z/OS, `DDNAME ETBCREP` is required to assign an output file for this report. Any failure will trigger a diagnostic message in the Broker log.

Load Module Report

The Load Module Report is created during startup of EntireX Broker on z/OS. All modules in all data sets concatenated to the `STEPLIB` chain for Broker execution are listed.

```
Operating System: z/OS 06.00
Node Name:        DAEF
IPL Date:         2007-10-02
IPL Time:         07:19:21
CPU Model:        2096

EntireX 8.0.0.12     Load Module Report 2007-10-02 08:56:23    Page     1
Total  Module   Date       Time  VRSPP Build number     Alias Level CurNo
                Steplib level 0: SAG.EXB731.LOAD
    1  ADAACK                                            NO      0     1
    2  ADABSP                                            NO      0     2
    3  ADACDC                                            NO      0     3
    4  ADACLU                                            NO      0     4
    5  ADACLX                                            NO      0     5
    6  ADACMO                                            NO      0     6
    7  ADACMP                                            NO      0     7
    8  ADACMR                                            NO      0     8
    9  ADACMU                                            NO      0     9
   10  ADACNS                                            NO      0    10
   11  ADACNV                                            NO      0    11
...
  156  ETBCMD   2007-10-01 15.48 73012 2007-10-01 15:01   NO      0   156
  157  ETBINFO  2007-10-01 15.48 73012 2007-10-01 15:01   NO      0   157
  158  ETBMISC                                           NO      0   158
  159  ETBNATTR 2007-10-01 15.48 73012 2007-10-01 15:01   NO      0   159
  160  ETBNUC   2007-10-01 15.48 73012 2007-10-01 15:01   NO      0   160
```

This report provides STEPLIB level, date, and time stamps if a certain pattern is used for the module structure. `DDNAME ETBMREP` must be assigned to get this report.

Storage Report

You can create an optional report file that provides details about all activities to allocate or to deallocate memory pools. This section details how to create the report and provides a sample report.

See also Broker-specific attribute `STORAGE-REPORT`.

Creating a Storage Report

Use Broker's global attribute `STORAGE-REPORT` with the value `YES`. If attribute value `YES` is supplied, all memory pool operations will be reported if the output mechanism is available. If the value `NO` is specified, no report will be created.

Platform-specific Rules

z/OS

`DDNAME ETBSREP` assigns the report file. Format `RECFM=FB, LRECL=121` is used.

UNIX and Windows

Broker creates a file with the name *STORAGE.REPORT* in the current working directory. If the environment variable `ETB_STORAGE_REPORT` is supplied, the file name specified in the environment variable will be used. If Broker receives the command-line argument `-r`, the token following argument `-r` will be used as the file name.

BS2000

`LINK-NAME ETBSREP` assigns the report file. Format `REC-FORM=V, REC-SIZE=0, FILE-TYPE ISAM` is used by default.

z/VSE

Logical unit `SYS015` and logical file name *ETBSREP* are used. Format `RECORD-FORMAT=FB, RECORD-LENGTH=121` is used.

Sample Storage Report

The following is an excerpt from a sample STORAGE report.

```
EntireX 8.1.0.00     STORAGE Report     2009-06-26 12:28:58    Page     1          ↩
                                                                                   ↩
                                                                                   ↩
Identifier                  Address            Size             Total          Date ↩
      Time      Action
KERNEL POOL                0x25E48010     407184 bytes     407184 bytes  2009-06-26 ↩
12:28:58.768  Allocated
HEAP POOL                  0x25EB4010    1050692 bytes    1457876 bytes  2009-06-26 ↩
12:28:58.769  Allocated
COMMUNICATION POOL         0x25FB5010   16781380 bytes   18239256 bytes  2009-06-26 ↩
12:28:58.769  Allocated
ACCOUNTING POOL            0x26FB7010     762052 bytes   19001308 bytes  2009-06-26 ↩
12:28:58.769  Allocated
BROKER POOL                0x27072010      61540 bytes   19062848 bytes  2009-06-26 ↩
12:28:58.775  Allocated
CONVERSATION POOL          0x27082010     368964 bytes   19431812 bytes  2009-06-26 ↩
12:28:58.775  Allocated
CONNECTION POOL            0x270DD010     233668 bytes   19665480 bytes  2009-06-26 ↩
12:28:58.779  Allocated
LONG MESSAGES POOL         0x27117010    4395204 bytes   24060684 bytes  2009-06-26 ↩
12:28:58.782  Allocated
SHORT MESSAGES POOL        0x27549010    3703876 bytes   27764560 bytes  2009-06-26 ↩
12:28:58.806  Allocated
PARTICIPANT POOL           0x278D2010     134244 bytes   27898804 bytes  2009-06-26 ↩
12:28:58.827  Allocated
PARTICIPANT EXTENSION POOL  0x278F3010      36996 bytes   27935800 bytes  2009-06-26 ↩
12:28:58.829  Allocated
PROXY QUEUE POOL           0x278FD010      26724 bytes   27962524 bytes  2009-06-26 ↩
12:28:58.829  Allocated
SERVICE ATTRIBUTES POOL     0x27904010     131668 bytes   28094192 bytes  2009-06-26 ↩
12:28:58.829  Allocated
SERVICE POOL               0x27925010      54372 bytes   28148564 bytes  2009-06-26 ↩
12:28:58.830  Allocated
SERVICE EXTENSION POOL      0x27933010      32900 bytes   28181464 bytes  2009-06-26 ↩
12:28:58.831  Allocated
TIMEOUT QUEUE POOL         0x2793C010      87268 bytes   28268732 bytes  2009-06-26 ↩
12:28:58.831  Allocated
TRANSLATION POOL           0x27952010     179300 bytes   28448032 bytes  2009-06-26 ↩
12:28:58.832  Allocated
UNIT OF WORK POOL          0x2797E010     176324 bytes   28624356 bytes  2009-06-26 ↩
12:28:58.834  Allocated
WORK QUEUE POOL            0x279AA010     391268 bytes   29015624 bytes  2009-06-26 ↩
12:28:58.835  Allocated
BLACKLIST POOL             0x27A0A010      42084 bytes   29057708 bytes  2009-06-26 ↩
12:28:58.838  Allocated
COMMUNICATION POOL         0x25FB5010   16781380 bytes   12837332 bytes  2009-06-26 ↩
```

```
12:30:58.514  Deallocated
ACCOUNTING POOL           0x26FB7010    762052 bytes   12075280 bytes  2009-06-26 ↩
12:30:58.515  Deallocated
BROKER POOL               0x27072010     61540 bytes   12013740 bytes  2009-06-26 ↩
12:30:58.516  Deallocated
CONVERSATION POOL         0x27082010    368964 bytes   11644776 bytes  2009-06-26 ↩
12:30:58.518  Deallocated
CONNECTION POOL           0x270DD010    233668 bytes   11411108 bytes  2009-06-26 ↩
12:30:58.519  Deallocated
LONG MESSAGES POOL        0x27117010   4395204 bytes    7015904 bytes  2009-06-26 ↩
12:30:58.520  Deallocated
SHORT MESSAGES POOL       0x27549010   3703876 bytes    3312028 bytes  2009-06-26 ↩
12:30:58.526  Deallocated
PROXY QUEUE POOL          0x278FD010     26724 bytes    3285304 bytes  2009-06-26 ↩
12:30:58.530  Deallocated
TIMEOUT QUEUE POOL        0x2793C010     87268 bytes    2690464 bytes  2009-06-26 ↩
12:30:58.532  Deallocated
UNIT OF WORK POOL         0x2797E010    176324 bytes    2514140 bytes  2009-06-26 ↩
12:30:58.533  Deallocated
WORK QUEUE POOL           0x279AA010    391268 bytes    2122872 bytes  2009-06-26 ↩
12:30:58.533  Deallocated
BLACKLIST POOL            0x27A0A010     42084 bytes    2080788 bytes  2009-06-26 ↩
12:30:58.534  Deallocated
PARTICIPANT POOL          0x278D2010    134244 bytes    1893112 bytes  2009-06-26 ↩
12:49:25.817  Deallocated
PARTICIPANT EXTENSION POOL  0x278F3010     36996 bytes    1856116 bytes  2009-06-26 ↩
12:49:25.818  Deallocated
SERVICE ATTRIBUTES POOL    0x27904010    131668 bytes    1724448 bytes  2009-06-26 ↩
12:49:25.818  Deallocated
SERVICE POOL              0x27925010     54372 bytes    1670076 bytes  2009-06-26 ↩
12:49:25.818  Deallocated
SERVICE EXTENSION POOL     0x27933010     32900 bytes    1637176 bytes  2009-06-26 ↩
12:49:25.819  Deallocated
TRANSLATION POOL          0x27952010    179300 bytes    1457876 bytes  2009-06-26 ↩
12:49:25.819  Deallocated
HEAP POOL                0x25EB4010   1050692 bytes     407184 bytes  2009-06-26 ↩
12:49:25.820  Deallocated
KERNEL POOL               0x25E48010    407184 bytes          0 bytes  2009-06-26 ↩
12:49:25.820  Deallocated
```

Header	Description
Identifier	Name of the memory pool.
Address	Start address of the memory pool.
Size	Size of the memory pool.
Total	Total size of all obtained memory pools.
Date, Time	Date and time of the action.
Action	The action of Broker. The following actions are currently supported: Allocated: memory pool is allocated . Deallocated: memory pool is deallocated.

Persistent Store Report

You can create an optional report file that provides details about all records added to or deleted from the persistent store. This section details how to create the report and provides a sample report.

Configuration

To create a persistent store report, use Broker's global attribute `PSTORE-REPORT` with the value `YES`.

When the attribute value `YES` is supplied, all created or deleted persistent records will be reported if the output mechanism is available.

If the value `NO` is specified, no report will be created.

The report file is created using the following rules:

BS2000

`LINK-NAME ETBPREP` assigns the report file. Format `REC-FORM=V, REC-SIZE=0, FILE-TYPE ISAM` is used by default.

UNIX

Broker creates a file with the name *PSTORE.REPORT* in the current working directory. The file name *PSTORE.REPORT.LOAD* will be used if Broker is started with `RUN-MODE` = `PSTORE-LOAD`.

The file name *PSTORE.LOAD.UNLOAD* will be used if Broker is started with `RUN-MODE` = `PSTORE-UNLOAD`.

If the environment variable `ETB_PSTORE_REPORT` is supplied, the file name specified in the environment variable will be used.

If Broker receives the command-line argument `-p`, the token following argument `-p` will be used as the file name.

Windows

Same as UNIX.

z/OS

`DDNAME ETBPREP` assigns the report file. Format `RECFM=FB, LRECL=121` is used.

z/VSE

Logical unit `SYS003` and logical file name *ETBPREP* are used. Format `RECORD-FORMAT` = `FB`, `RECORD-LENGTH` = `121` is used.

Sample Report

The following is an excerpt from a sample `PSTORE` report.

```
EntireX 8.0.0.00     PSTORE Report      2008-02-21 17:18:38    Page     1

Identifier        Elements  Total length  Record Type   Date       Time          ↩
Action
1000000000D000016        5          1148   Conversation  2008-02-21 17:18:57.190  ↩
Created
1000000000D000017        5          1148   Conversation  2008-02-21 17:18:57.654  ↩
Created
1000000000D000018        5          1148   Conversation  2008-02-21 17:18:58.122  ↩
Created
1000000000D000019        5          1148   Conversation  2008-02-21 17:18:58.590  ↩
Created
1000000000D00001A        5          1148   Conversation  2008-02-21 17:18:59.054  ↩
Created
1000000000D00001B        5          1148   Conversation  2008-02-21 17:18:59.518  ↩
Created
1000000000D00001C        5          1148   Conversation  2008-02-21 17:18:59.982  ↩
Created
1000000000D00001D        5          1148   Conversation  2008-02-21 17:19:00.538  ↩
Created
1000000000D00001E        5          1148   Conversation  2008-02-21 17:19:01.002  ↩
Created
1000000000C000001        0             0   Conversation  2008-02-21 17:19:30.676  ↩
Deleted
1000000000C000002        0             0   Conversation  2008-02-21 17:19:31.675  ↩
Deleted
1000000000C000003        0             0   Conversation  2008-02-21 17:19:32.675  ↩
Deleted
1000000000C000004        0             0   Conversation  2008-02-21 17:19:33.675  ↩
Deleted
1000000000C000005        0             0   Conversation  2008-02-21 17:19:34.675  ↩
Deleted
1000000000C000006        0             0   Conversation  2008-02-21 17:19:35.675  ↩
Deleted
```

The following fields are provided in the report:

- `Identifier` provides the UOWID (record ID).
- `Elements` gives the number of messages per UOW when creating or loading records.
- `Total Length` gives the size of the raw record when creating or loading records.

- `Record Type` describes the type of the data. Following types are currently supported:
  - `Cluster`: a special record for synchronization purposes
  - `Conversation`: a unit of work as part of a conversation
  - `Master`: a special record to manage the persistent store
- Date and time of the action
- `Action` describes the action of Broker. The following actions are currently supported:
  - `Created`: record is created
  - `Deleted`: record is deleted
  - `Loaded`: record is loaded (Broker instance with `RUN-MODE` = `PSTORE-LOAD`)
  - `Unloaded`: record is unloaded (Broker instance with `RUN-MODE` = `PSTORE-UNLOAD`)

License Report

The License Report is created during broker startup on the respective platform. It contains the contents of the license file itself and some machine data.

z/OS

DDNAME `ETBLREP` must be assigned to get this report. See *Step 2: Edit the Broker Startup Procedure*.

BS2000

`LINK-NAME ETBLREP` must be assigned to get this report.

13 Command Logging in EntireX

Command logging is a feature to assist in debugging Broker ACI applications. A command in this context represents one user request sent to the Broker and the related response of Broker.

Command logging is a feature that writes the user requests and responses to file in a way it is already known with Broker trace and `TRACE-LEVEL=1`. But command logging works completely independent from trace, and data is written to a file only if defined command trace filters detect a match.

Broker stub applications send commands or requests to the Broker kernel, and the Broker kernel returns a response to the requesting application. Developers who need to resolve problems in an application need access to those request and response strings inside the Broker kernel. That's where command logging comes in. With command logging, request and response strings from or to an application are written to a file that is separate from the Broker trace file.

Introduction to Command Logging

This section provides an introduction to command logging in EntireX and offers examples of how command logging is implemented. It covers the following topics:

- Overview
- Command Log Files
- Defining Filters
- Programmatically Turning on Command Logging

Overview

Command logging is similar to a Broker trace that is generated when the Broker attribute `TRACE-LEVEL` is set to 1. Broker trace and command logging are independent of each other, and therefore the configuration of command logging is separate from Broker tracing.

The following Broker attributes are involved in command logging:

Attribute	Description
`CMDLOG`	Set this to "N" if command logging is not needed.
`CMDLOG-FILE-SIZE`	A numeric value indicating the maximum size of command log file in KB.
`NUM-CMDLOG-FILTER`	The maximum number of filters that can be set.

In addition to `CMDLOG=YES`, the Broker needs the assignment of the dual command logging files during startup. If these assignments are missing, Broker will set `CMDLOG=NO`. See also *Broker Attributes*.

Command Log Files

The Broker keeps a record of commands (request and response strings) in a command log file.

At Broker startup, you will need to supply two command log file names and paths. Only one file is open at a time, however, and the Broker writes commands (requests and responses) to this file.

Under UNIX and Windows, the startup options `-y` and `-z` are evaluated by executable `etbnuc`. These options are used to specify the command log file names. Startup script/service assign these files by default.

Under z/OS, the file requirements are two equally sized, physical sequential files defined with a record length of 121 bytes, i.e. `DCB=(LRECL=121,RECFM=FB,BLKSIZE=nnnn)`. We recommend you allocate files with a single (primary) extent only. For example `SPACE=(CYL,(30,0))`. The minimum file size is approximately 3 cylinders of 3390 device. Alternatively, the dual command log files can be allowed in USS HFS file system.

When the size of the active command log file reaches the KB limit set by `CMDLOG-FILE-SIZE`, the file is closed and the second file is opened and becomes active. When the second file also reaches the KB limit set by `CMDLOG-FILE-SIZE`, the first file is opened and second file is closed. Existing log data in a newly opened file will be lost.

Defining Filters

In command logging, a filter is used to store and identify a class, server, or service, as well as a user ID.

Use the command-line tool `etbcmd` to define a filter. During processing, the Broker evaluates the class, server, service, and user ID associated with each incoming request and compares them with the same parameters specified in the filters. If there is a match, the request string and response string of the request is printed out to the command log file.

Programmatically Turning on Command Logging

Applications using ACI version 9 or above have access to the new field `LOG-COMMAND` in the ACI control block.

If this field is set, the accompanying request and the Broker's response to this request is logged to the command log file.

Note: Programmatic command logging ignores any filters set in the kernel.

Command Log Filtering using Command-line Interface ETBCMD

The examples assume that Broker has been started with the attribute `CMDLOG=Y`.

Setting Filters

Filters need to be set before running the stub applications whose commands are to be logged.

UNIX and Windows

Command	Description
`etbcmd -blocalhost:1970:TCP -cSET-CMDLOG-FILTER -dBROKER -xuser -nACLASS/ASERVER/ASERVICE`	This command sets filters on `ACLASS/ASERVER/ASERVICE`. All ACI calls issued by *all* users to this service will be logged.
`etbcmd -blocalhost:1970:TCP -cSET-CMDLOG-FILTER -dBROKER -xuser -nACLASS/ASERVER/ASERVICE -Usaguser1`	This command set filters on `ACLASS/ASERVER/ASERVICE` and user ID `saguser1`. All ACI calls to this service *as well as* those issued by `saguser1` will be logged.

z/OS

Command	Description

`//ETBCMD  EXEC PGM=ETBCMD,`
`// PARM=('/-blocalhost:1970:TCP ↩`
`-cSET-CMDLOG-FILTER -xuser ',`
`//       '-dBROKER ↩`
`-nACLASS/ASERVER/ASERVICE')`
	This command sets filters on `ACLASS/ASERVER/ASERVICE`. All ACI calls issued by *all* users to this service will be logged.

`//ETBCMD  EXEC PGM=ETBCMD,`
`// PARM=('/-blocalhost:1970:TCP ↩`
`-cSET-CMDLOG-FILTER -xuser ',`
`//      '-dBROKER -nACLASS/ASERVER/ASERVICE ↩`
`-Usaguser1')`
	This command sets filters on `ACLASS/ASERVER/ASERVICE` and user ID `saguser1`. All ACI calls to this service *as well as* those issued by `saguser1` will be logged.

Note: If more than one service is set as a filter, all ACI calls sent to any of these services will be logged. Identical filters cannot be set. Attempts to set a second filter that matches an existing filter will be rejected. Similarly, the maximum number of filters that can be added is defined in `NUM-CMDLOG-FILTER`. If the maximum number of filters is already being used, delete an existing filter to make room for a new filter.

Deleting Filters

The following provides an example of how to delete an existing filter on a service.

To delete a filter

- Enter the following command.

  Under UNIX:

```
etbcmd -d BROKER -b localhost:1970:TCP -c CLEAR-CMDLOG-FILTER ↩
-nACLASS/ASERVER/ASERVICE -U saguser1
```

  Under z/OS:

```
//ETBCMD  EXEC PGM=ETBCMD,
// PARM=('/-blocalhost:1970:TCP -cCLEAR-CMDLOG-FILTER -xuser ',
//       '-dBROKER -nACLASS/ASERVER/ASERVICE')
```

  If the filter does not exist, the command will return an error.

Disabling and Enabling a Filter

Filters can be set and still be disabled (made inactive).

To disable a filter

- Enter the following command.

  Under UNIX:

```
etbcmd -blocalhost:1970:TCP -cDISABLE-CMDLOG-FILTER -dBROKER -xuser ↩
-nACLASS/ASERVER/ASERVICE -Usaguser1
```

  Under z/OS:

```
//ETBCMD  EXEC PGM=ETBCMD,
// PARM=('/-blocalhost:1970:TCP -cDISABLE-CMDLOG-FILTER -xuser ',
//       '-dBROKER -nACLASS/ASERVER/ASERVICE -Usaguser1')
```

  Note: A disabled filter will not bring down the count of filters in use.

To enable a filter

- Enter the following command to enable the disabled filter.

Under UNIX:

```
etbcmd -blocalhost:1970:TCP -cENABLE-CMDLOG-FILTER -dBROKER -xuser ↩
-nACLASS/ASERVER/ASERVICE -Usaguser1
```

Under z/OS:

```
//ETBCMD  EXEC PGM=ETBCMD,
// PARM=('/-blocalhost:1970:TCP -cENABLE-CMDLOG-FILTER -xuser ',
//       '-dBROKER -nACLASS/ASERVER/ASERVICE -Usaguser1')
```

ACI-driven Command Logging

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

When handling ACI functions with command log turned on, Broker will not evaluate any filters. Application developers must remember to reset the `LOG-COMMAND` field if subsequent requests are not required to be logged.

Dual Command Log Files

Broker's use of two command log files prevents any one command log file from becoming too large.

When starting a Broker with command log support, you must therefore specify two file names and paths - one for each of the two command log files. The sample startup script installed with the product uses the variables `ETB_CMDLOG1` and `ETB_CMDLOG2` as the default command log file names.

Under UNIX, the startup script uses file names `CMDLOGR1` and `CMDLOGR2`.

Under Windows, the keys `ETB_CMDLOG1` and `ETB_CMDLOG2` are entered in the Registry with values `CMDLOGR1` and `CMDLOGR2`.

At startup, Broker initializes both files and keeps one of them open. Command log statements are printed to the open file until the size of this file reaches the value specified in the Broker attribute `CMDLOG-FILE-SIZE`. This value must be specified in KB.

When the size of the open file exceeds the value specified in the Broker attribute `CMDLOG-FILE-SIZE`, Broker closes this file and opens the other, dormant file. Because the Broker closes a log file

only when unable to print out a complete log line, the size of a *full* file may be smaller than `CMDLOG-FILE-SIZE`.

To switch log files on demand, using `etbcmd` | `ETBCMD`

- An open command log file can be forcibly closed even before the size limit is reached. Enter the following command.

  Under UNIX:

```
etbcmd -blocalhost:1970:TCP -cSWITCH-CMDLOG -dBROKER -xuser
```

  Under z/OS:

```
//ETBCMD  EXEC PGM=ETBCMD,
// PARM=('/-blocalhost:1970:TCP -cSWITCH-CMDLOG -xuser ',
//       '-dBROKER')
```

  The command above will close the currently open file and open the one that has been dormant.