webMethods EntireX

Administration under z/VSE

Version 9.6

April 2014

This document applies to webMethods EntireX Version 9.6.

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

Document ID: EXX-ADMIN-96-20140628VSE

Table of Contents

1 Setting up Broker Instances

This chapter contains information on setting up the Broker under z/VSE. It assumes that you have completed the relevant steps described under *Installing EntireX under z/VSE*.

Setting up TCP/IP Transport

The recommended way to set up the TCP/IP communicator is to define `PORT=nnnn` and optionally `HOST=x.x.x.x|host_name` under *TCP/IP-specific Attributes* (`DEFAULTS=TCP`) under *Broker Attributes* in the platform-independent administration documentation.

However, if no port number is specified in the broker attribute file, the broker kernel will use default port number of 1971. This is the same default port number that the stubs use.

Setting up Entire Net-Work/Adabas SVC Transport

To set up EntireX Net-Work communication mechanism

1 Ensure that appropriate values are supplied in the broker attribute file section `DEFAULTS=NET`, paying particular attention to the `IUBL` parameter - which specifies the maximum send/receive buffer length that can be sent between an application and Broker kernel within a single request - and `NABS`, which governs the total amount of storage available concurrently for all users communicating over this transport mechanism. See *Adabas SVC/Entire Net-Work-specific Attributes* (`DEFAULTS=NET`) under *Broker Attributes* in the platform-independent administration documentation.

2 Ensure that communication with the broker is possible by running the installation verification programs (`BCOC`, `BCOS`) using transport type NET.

Starting and Stopping the Broker

Starting the Broker

To start the broker

- Run job `RUNETB.J`.

  If the UPSI bit is not set (see Broker installation step *Step 4: Customize the EntireX Broker Startup Job Control (`RUNETB.J`)*) you will be prompted on the console with following message: `Broker V9.6.0.00 ready for communication: ETB00001`.

Stopping the Broker

To stop the broker

- Use the following console command:

```
task_id  ETBSTOP
```

If the console prompt is suppressed, enter an MSG command before the console command:

```
MSG partition_id
```

Tracing EntireX Broker

This section covers the following topics:

Broker TRACE-LEVEL Attribute

The Broker `TRACE-LEVEL` attribute determines the level of tracing to be performed while Broker is running. The Broker has a master `TRACE-LEVEL` specified in the Broker section of the attribute file as well as several individual `TRACE-LEVEL` settings that are specified in the following sections of the attribute file. You can also modify the different `TRACE-LEVEL` values while Broker is running, without having to restart the Broker kernel for the change to take effect.

Individual Settings	Specified in Attribute File Section
Master trace level	`DEFAULTS=BROKER`
Conversion trace level	Trace option of the `CONVERSION` parameter that can be defined in `DEFAULTS=SERVICE | TOPIC`
Security trace level	`DEFAULTS=SECURITY`
Transport trace level	`DEFAULTS=NET | TCP`

These individual `TRACE-LEVEL` values determine the level of tracing within each subcomponent. If not specified, the master `TRACE-LEVEL` is used.

Attribute File Trace Setting

Trace Level	Description
0	No tracing. Default value.
1	Traces incoming requests, outgoing replies, and resource usage.
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.

Note: Trace levels 2 and above should be used only when requested by Software AG support.

Deferred Tracing

It is not always convenient to run with `TRACE-LEVEL` defined, especially when higher trace levels are involved. Deferred tracing is triggered when a specific condition occurs, such as an ACI response code or a broker subtask abend. Such conditions cause the contents of the trace buffer to be written, showing trace information leading up the specified event. If the specified event does not occur, the Broker trace will contain only startup and shutdown information (equivalent to `TRACE-LEVEL=0`). Operating the trace in this mode requires the following additional attributes in the broker section of the attribute file. Values for `TRBUFNUM` and `TRAP-ERROR` are only examples.

Attribute	Value	Description
`TRBUFNUM`	3	Specifies the deferred trace buffer size = 3 * 64 K.
`TRMODE`	WRAP	Indicates trace is not written until an event occurs.
`TRAP-ERROR`	322	Assigns the event ACI response code 00780322 "PSI: UPDATE failed".

Protecting a Broker against Denial-of-Service Attacks

An optional feature of EntireX Broker is available to protect a broker running with `SECURITY=YES` against denial-of-service attacks. An application that is running with invalid user credentials will get a security response code. However, if the process is doing this in a processing loop, the whole system could be affected. If `PARTICIPANT-BLACKLIST` is set to `YES`, EntireX Broker maintains a blacklist to handle such “attacks”. If an application causes ten consecutive security class error codes within 30 seconds, the blacklist handler puts the participant on the blacklist. All subsequent requests from this participant are blocked until the `BLACKLIST-PENALTY-TIME` has elapsed.

Server Shutdown Use Case

Here is a scenario illustrating another use of this feature that is not security-related.

An RPC server is to be shut down immediately, using Broker Command and Information Services (CIS), and has no active request in the broker. The shutdown results in the `LOGOFF` of the server. The next request that the server receives will probably result in message 00020002 "User does not exist", which will cause the server to reinitialize itself. It was not possible to inform the server that shutdown was meant to be performed.

With the *blacklist*, this is now possible. As long as the blacklist is not switched off, when a server is shut down immediately using CIS and when there is no active request in the broker, a marker is set in the blacklist. When the next request is received, this marker results in message 00100050 "Shutdown IMMED required", which means that the server is always informed of the shutdown.

2 Administration of Broker Stubs

Available Stubs

This table lists all Broker stubs available under the z/VSE operating system that are to be used with the programming languages Natural, COBOL, Assembler and C.

The stub you choose depends on the following:

- the environment (CICS or Batch)
- the availability of administration features such as tracing and compression

Environment	Transport			Trace	Stub Module
	NET	TCP	SSL		
All environments that use Batch	Yes	Yes	No	Yes*	*BKIMB*
All environments that use CICS	Yes	Yes	No	Yes*	*BKIMC*

* The request needs to use TCP transport method. Tracing is not available with NET transport.

Transport Methods for Broker Stubs

Transport Method Values

Transport Value	Tips
TCP	■ Provides remote machine and cross-platform communication.
NET	■ Provides the best performing transport if the application and Broker kernel reside on the same machine. ■ Provides for remote communications if Entire Net-Work is also installed on the application and Broker kernel machines. ■ Requires the installation of Adabas components. We recommend installing the Adabas modules delivered with EntireX installation kit. See your Adabas documentation for more information on installing the Adabas SVC. ■ Tracing is not supported.

To use the stubs' internal security functionality, API version 8 or higher needs to be used by the application. (e.g., EntireX RPC Server, NAT42). The delivered phases are linked for use with internal security.

Setting the Timeout for the Transport Method

Introduction

If the transport layer is interrupted, communication between the broker and the stub - that is, client or server application - is no longer possible. A client or server might possibly wait infinitely for a broker reply or message in such a situation. To prevent this and return control to your calling application in such a situation, set a timeout value for the transport method.

The timeout settings for transport layers are independent of the timeout settings of the broker.

Setting the timeout for the transport layer is possible for the transport method TCP, and is supported by all broker stubs under z/VSE.

Transport Timeout Values

The timeout value for the transport method is set by the environment variable `TIMEOUT` on the stub side. This transport timeout is used together with the broker timeout - which is set by the application in the `WAIT` field of the broker ACI control block - to calculate the actual value for the transport layer's timeout. The following table describes the possible values for the transport timeout:

Transport Timeout Value	Description
0	Infinite wait for the application.
n	The transport method additionally waits this time in seconds. A negative value is treated as `TIMEOUT=0` (infinite wait for the application).
nothing set	Transport method waits additional 20 seconds.

The actual timeout for transport layer equals broker timeout (`WAIT` field) + timeout value for transport method.

Limiting the TCP/IP Connection Lifetime

With transport method TCP/IP, the broker stub establishes one or more TCP/IP connections to the brokers specified with `BROKER-ID`. These connections can be controlled by the transport-specific `CONNECTION-NONACT` attribute on the broker side, but also by the transport-specific environment variable `NONACT` on the stub side. If `NONACT` is not 0, it defines the non-activity time (in seconds) of active TCP/IP connections to any broker. See `NONACT` under *Environment Variables in EntireX*. Whenever the broker stub is called, it checks for the elapsed non-activity time and closes connections with a non-activity time greater than the value defined with `NONACT`.

Transport Non-activity Value	Description
0	Infinite lifetime until application is stopped.
n (seconds)	Transport connections with non-activity time greater than *n* will be closed.
Nothing set	Transport connections with non-activity time greater than 300s (default) will be closed.

Using the Batch Stub Interface Module BKIMB

You can use BKIMB for all batch environments. This stub interface module is delivered as a phase, which can be loaded by your application dynamically, and as an object for linking. During runtime, the EXX960 library and the WAL826 library need to be included into the LIBDEF search chain. If you need to statically link your application with the interface object, include the following objects:

```
PHASE <appl_phase_name>,*
INCLUDE <app_obj>
INCLUDE BKIMB
INCLUDE ETBVPRE
INCLUDE ETBVEVA
INCLUDE ETBENC
INCLUDE ETBTB
ENTRY <app_entry>
```

To set up a secure environment

1 Statically link your application with the following interface objects:

```
PHASE <appl_phase_name>,*
INCLUDE <app_obj>
INCLUDE BKIMB
INCLUDE ETBUPRE
INCLUDE ETBUEVA
INCLUDE ETBVPRE
INCLUDE ETBVEVA
INCLUDE ETBENC
INCLUDE ETBTB
ENTRY <app_entry>
```

Or:

If BKIMB is to be loaded dynamically, you can relink the phase for use with security. Refer to the delivered job control example BKIMB.J.

2 Rename phase SECUEXI0 in library EXX960 to SECUEXIT.

Using the CICS Stub Interface Module BKIMC

You can use BKIMC for all CICS environments. This stub interface module is delivered as a phase, which can be loaded by your application dynamically, and as an object for linking. To enable CICS to find the various programs, include the EXX960 sublibrary in the DFHRPL chain and add following definition to your CICS environment:

```
DEFINE PROGRAM(BKIMC) GROUP(EXX) LANGUAGE(ASSEMBLER)  (only required if not linked ↩
to your application)
DEFINE PROGRAM(BROKERC) GROUP(EXX) LANGUAGE(C)
```

If you need to statically link your application with the interface object, include the following objects:

```
PHASE <appl_phase_name>,*
INCLUDE <app_obj>
INCLUDE BKIMC
INCLUDE ETBVPRE
INCLUDE ETBVEVA
INCLUDE ETBENC
INCLUDE ETBTB
ENTRY <app_entry>
```

To set up a secure environment

1 Statically link your application with the following interface objects:

```
PHASE <appl_phase_name>,*
INCLUDE <app_obj>
INCLUDE BKIMC
INCLUDE ETBUPRE
INCLUDE ETBUEVA
INCLUDE ETBVPRE
INCLUDE ETBVEVA
INCLUDE ETBENC
INCLUDE ETBTB
ENTRY <app_entry>
```

Or:

If BKIMC is to be loaded dynamically, you can relink the phase for use with security. Refer to the delivered job control example BKIMC.J.

2 Rename phase SECUEXI0 in library EXX960 to SECUEXIT.

Data Encryption

Broker control block field `ENCRYPTION-LEVEL` and broker attribute `ENCRYPTION-LEVEL` determine whether data encryption is used. The password is always encrypted; user IDs are not encrypted.

Tracing for Broker Stubs

If transport method TCP is used, a stub trace may be turned on for diagnostic purposes. Set up the following environment variable in your application job control or CICS startup.

```
//SETPARM STUBLOG=2
```

3 Operator Commands

Command Syntax

The following command format is required to communicate with EntireX Broker, using the operator console. Parameters in UPPERCASE must be typed "as is". Parameters in lowercase must be substituted with a valid value. Operator commands have the following format:

```
task_id command[parameter]
```

where	*task_id*	is the ID of the broker task
	command	is the operator command
	parameter	is an optional parameter allowed by the operator command you are issuing

General Broker Commands

The following broker commands are available:

- BROKER TRACE
- DPOOL
- DRES
- DSTAT
- ETBEND
- ETBSTOP
- FLUSH
- PSTORE TRACE
- SHUTDOWN class,server,service
- TRACE
- TRAP-ERROR

BROKER TRACE

Alias of broker command `TRACE`. Modifies the setting of the broker-specific attribute `TRACE-LEVEL`.

Example

To set a trace level 2 for broker

- Enter command:

```
task_id BROKER TRACE=2
```

If the console prompt is suppressed, enter an `MSG` command before the console command:

```
MSG partition_id
```

See `TRACE-LEVEL` under *Broker Attributes* in the platform-independent administration documentation.

DPOOL

Lists all memory pools currently allocated by EntireX Broker. Start address, pool size in bytes and name of pool are provided. There can be multiple entries for a specific type of pool.

Sample Output

```
ETBM0720 Operator typed in: DPOOL
ETBM0657 Broker pool usage:
ETBM0657 0x2338FFB8   16781380 bytes COMMUNICATION POOL
ETBM0657 0x243A9EB8     368964 bytes CONVERSATION POOL
ETBM0657 0x24404F38     233668 bytes CONNECTION POOL
ETBM0657 0x2443EF38    4395204 bytes LONG MESSAGES POOL
ETBM0657 0x24870BB8    3703876 bytes SHORT MESSAGES POOL
ETBM0657 0x24BF9398     134244 bytes PARTICIPANT POOL
ETBM0657 0x24C1AF78      36996 bytes PARTICIPANT EXTENSION POOL
ETBM0657 0x24C24798      26724 bytes PROXY QUEUE POOL
ETBM0657 0x24C2BDA8     131668 bytes SERVICE ATTRIBUTES POOL
ETBM0657 0x24C4CB98      54372 bytes SERVICE POOL
ETBM0657 0x24C5AF78      32900 bytes SERVICE EXTENSION POOL
ETBM0657 0x24D31FA8     344148 bytes SUBSCRIPTION POOL
ETBM0657 0x24D865A8     129620 bytes TOPIC ATTRIBUTES POOL
ETBM0657 0x2338F420       2952 bytes TOPIC POOL
ETBM0657 0x24DA6778      30852 bytes TOPIC EXTENSION POOL
ETBM0657 0x24C63B18      87268 bytes TIMEOUT QUEUE POOL
ETBM0657 0x24C79398     179300 bytes TRANSLATION POOL
ETBM0657 0x24CA5F38     176324 bytes UNIT OF WORK POOL
ETBM0657 0x24CD1798     391268 bytes WORK QUEUE POOL
ETBM0657 0x24DAEB98      33892 bytes PSTORE SUBSCRIBER POOL
ETBM0657 0x24DB73A8      19540 bytes PSTORE TOPIC POOL
ETBM0582 Function completed
```

DRES

Displays EntireX Broker's resource usage for conversations, message buffers, participants, services, topics, the timeout queue, units of work, and the work queue. Resource usage provides the total number, the number of free elements, and the number of used elements.

Sample Output

```
ETBM0720 Operator typed in: DRES
ETBM0581 Broker resource usage:
ETBM0581 Resource ----------------- Total # --- Free # --- Used #
ETBM0581 Conversations                 4096        852       3244
ETBM0581 Long message buffers             0          0          0
ETBM0581 Short message buffers         8192       7384        808
ETBM0581 Participants                   256        235         21
ETBM0581 Services                       256        240         16
ETBM0581 Topics                           0          0          0
ETBM0581 Timeout Queue                 1280        845        435
ETBM0581 Units Of Work                    0          0          0
ETBM0581 Work Queue                     256        239         17
ETBM0582 Function completed
```

DSTAT

Displays the total number of active elements, and an optional high watermark for services, clients, servers, conversations, message buffers, topics, publishers, subscribers, and publications.

Sample Output

```
ETBM0720 Operator typed in: DSTAT
ETBM0580 Broker statistics:
ETBM0580 NUM-SERVICE ............... 0
ETBM0580 Services active ........... 7
ETBM0580 NUM-CLIENT ................ 0
ETBM0580 Clients active ............ 10
ETBM0580 Clients active HWM ........ 10
ETBM0580 NUM-SERVER ................ 0
ETBM0580 Servers active ............ 10
ETBM0580 Servers active HWM ........ 10
ETBM0580 NUM-CONVERSATION .......... 0
ETBM0580 Conversations active ...... 607
ETBM0580 Conversations active HWM .. 968
ETBM0580 NUM-LONG-BUFFER ........... 0
ETBM0580 Long buffers active ....... 0
ETBM0580 Long buffers active HWM ... 0
ETBM0580 NUM-SHORT-BUFFER .......... 0
ETBM0580 Short buffers active ...... 1219
ETBM0580 Short buffers active HWM .. 1928
ETBM0580 NUM-TOPIC ................. 0
ETBM0580 Topics active ............. 0
```

```
ETBM0580 NUM-PUBLISHER ............. 0
ETBM0580 Publishers active ......... 0
ETBM0580 Publishers active HWM ..... 0
ETBM0580 NUM-SUBSCRIBER ............ 0
ETBM0580 Subscribers active ........ 0
ETBM0580 Subscribers active HWM .... 0
ETBM0580 NUM-PUBLICATION ........... 0
ETBM0580 Publications active ....... 0
ETBM0580 Publications active HWM ... 0
ETBM0582 Function completed
```

ETBEND

Processing stops immediately. Current calls to the EntireX Broker are not allowed to finish.

ETBSTOP

Alias of `ETBEND`.

FLUSH

Flush all trace data kept in internal trace buffers to stderr (`SYSOUT`). The broker-specific attribute `TRMODE=WRAP` is required.

PSTORE TRACE

Modifies the trace level for the Adabas persistent store (Adabas-specific attribute `TRACE-LEVEL`).

Example

To set a trace level 2 for the Adabas persistent store

- Enter command:

```
task_id PSTORE TRACE=2
```

See `TRACE-LEVEL` under *Broker Attributes* in the platform-independent administration documentation.

SHUTDOWN class,server,service

Shuts down the specified service immediately and stops all servers that have registered this service.

Example

To shutdown service `CLASS=RPC, SERVER=SRV1, SERVICE=CALLNAT`

- Enter command:

```
task_id SHUTDOWN RPC,SRV1,CALLNAT
```

TRACE

Modifies the setting of the broker-specific attribute `TRACE-LEVEL`.

Sample Commands

To modify the trace level

- Enter command, for example:

```
task_id TRACE=0
task_id TRACE=1
task_id TRACE=4
```

See `TRACE-LEVEL` under *Broker Attributes* in the platform-independent administration documentation.

TRAP-ERROR

Modifies the setting of the broker-specific attribute `TRAP-ERROR`.

Sample Command

To modify the setting for `TRAP-ERROR`

- Enter command:

```
task_id TRAP-ERROR=nnnn
```

where *nnnn* is the four-digit API error number that triggers the trace handler.

See TRAP-ERROR under *Broker Attributes* in the platform-independent administration documentation.

Participant-specific Commands

Within EntireX Broker nomenclature, a participant is an application implicitly or explicitly logged on to the Broker as a specific user. A participant could act as client, server, publisher or subscriber. The following participant-specific commands are available:

- CANCEL parameter
- USERLIST
- USERS parameter

CANCEL parameter

Operator command `CANCEL` is used to delete participants from EntireX Broker. The following parameters are supported:

Parameter	Description
`[USER=]`*`user_id`*	Cancel all participants with the specified *`user_id`*. Non-persistent resources will be freed by the timeout manager. Prefix "`USER=`" is the default value and may be omitted.
`SEQNO=`*`seqno`*	Cancel the participant with the sequence number *`seqno`*. Non-persistent resources will be freed by the timeout manager. Operator commands `USERLIST` and `USERS` display sequence numbers of all selected participants.

Sample Commands

To cancel all participant entries of user "DOE"

- Enter command:

```
task_id CANCEL DOE
```

Or:

```
task_id CANCEL USER=DOE
```

To cancel participant with sequence number "11"

- Enter command:

```
task_id CANCEL SEQNO=11
```

USERLIST

Operator command `USERLIST` displays a list of selected participant entries. The following parameters are supported:

Parameter	Description
none | *	Display all participants.
user_id	Display all participants with user ID *user_id*. Wildcard characters are supported.

Sample Commands

To display all participants

- Enter command:

```
task_id USERLIST
```

Or:

```
task_id USERLIST *
```

To display all participants with user ID "DOE"

- Enter command:

```
task_id USERLIST DOE
```

This produces the following output. See *Description of USERLIST Output Columns* below.

```
ETBM0720 Operator typed in: USERLIST DOE
ETBM0687 Participants:
ETBM0687 USER-ID ----------------------- C S P U E CHR SEQNO
ETBM0687 DOE                            N Y N N Y ASC 1
ETBM0582 Function completed
```

To display all participants with user ID starting with uppercase "D"

- Enter command:

```
task_id USERLIST D*
```

This produces the following output. See *Description of USERLIST Output Columns* below.

```
ETBM0720 Operator typed in: USERLIST D*
ETBM0687 Participants:
ETBM0687 USER-ID ------------------------ C S P U E CHR SEQNO
ETBM0687 DOE                             N Y N N Y ASC 1
ETBM0687 DOE1                            N Y N N Y EBC 2
ETBM0687 DOE2                            N Y N N Y EBC 3
ETBM0687 DOE3                            N Y N N Y EBC 4
ETBM0582 Function completed
```

To display all participants with 4-character user ID, starting with uppercase "D" and with uppercase "E" as third character

- Enter command:

```
task_id USERLIST D?E?
```

This produces the following output. See *Description of USERLIST Output Columns* below.

```
ETBM0720 Operator typed in: USERLIST D?E?
ETBM0687 Participants:
ETBM0687 USER-ID ------------------------ C S P U E CHR SEQNO
ETBM0687 DOE1                            N Y N N Y EBC 2
ETBM0687 DOE2                            N Y N N Y EBC 3
ETBM0687 DOE3                            N Y N N Y EBC 4
ETBM0582 Function completed
```

Description of USERLIST Output Columns

Keyword	Description
USER-ID	User ID (32 bytes, case-sensitive). See USER-ID under *Broker ACI Fields*.
C	Client. Y Participant is a client, otherwise "N".
S	Server. Y Participant is a server, otherwise "N".
P	Publisher. Y Participant is a publisher, otherwise "N".

Keyword	Description
U	Subscriber. Y Participant is a subscriber, otherwise "N".
E	Big endian. Y Participant is on a big-endian machine. N Participant is on a little-endian machine.
CHR	Character set. ASC Participant is an ASCII user. EBC Participant is an EBCDIC user.
SEQNO	Sequence number of participant. Can be used for operator command CANCEL *parameter*.

USERS parameter

Operator command USERS displays selected user data of participant entries. The following parameters are supported:

Parameter	Description
none | *	Display all participants.
user_id	Display all participants with user ID *user_id*. Wildcard characters are supported.

Sample Commands

To display all participants

- Enter command:

```
task_id USERS
```

Or:

```
task_id USERS *
```

To display all participants with user ID "DOE"

- Enter command:

```
task_id USERS DOE
```

This produces the following output. See ***Description of USERS Output Columns*** below.

```
ETBM0720 Operator typed in: USERS DOE
ETBM0687 Participants:
ETBM0687 USER-ID: DOE
ETBM0687 CLIENT: N SERVER: Y PUBLISHER: N SUBSCRIBER: N
ETBM0687 SEQNO: 6 BIG ENDIAN: Y CHARSET: ASCII PUID:
ETBM0687 202073756E6578322D2D303030303024646462D2D3030303030303031
ETBM0687 TOKEN:
ETBM0582 Function completed
```

Description of USERS Output Columns

Keyword	Description
USER-ID	User ID (32 bytes, case-sensitive). See USER-ID under *Broker ACI Fields*.
CLIENT	Y Participant is a client, otherwise "N".
SERVER	Y Participant is a server, otherwise "N".
PUBLISHER	Y Participant is a publisher, otherwise "N".
SUBSCRIBER	Y Participant is a subscriber, otherwise "N".
BIG ENDIAN	Y Participant is on a big-endian machine. N Participant is on a little-endian machine.
CHARSET	ASC Participant is an ASCII user. EBC Participant is an EBCDIC user.
PUID	Internal unique ID of participant. Hexadecimal 28-byte value in printable format.
TOKEN	Optionally identifies the participant. See TOKEN under *Broker ACI Fields*.

Security-specific Commands

DSECSTAT

Displays the number of successful and failed Security authentications and Security authorizations.

Sample Output

```
ETBM0720 Operator typed in: DSECSTAT
ETBM0579 Security Authentications - successful: 20 failed: 0
ETBM0579 Security Authorizations  - successful: 0 failed: 0
```

RESET userid

Resets the Security context for the specified user ID.

Sample Output

```
ETBM0720 Operator typed in: RESET EXXBATCH
ETBM0578 Reset ACEE for SAF-ID EXXBATCH : 20 instances found
```

SECURITY TRACE

Modifies the trace level for the EntireX Security (security-specific attribute `TRACE-LEVEL`). Broker-specific attribute `SECURITY=YES` must be set.

Example

To set a trace level 2 for EntireX Security

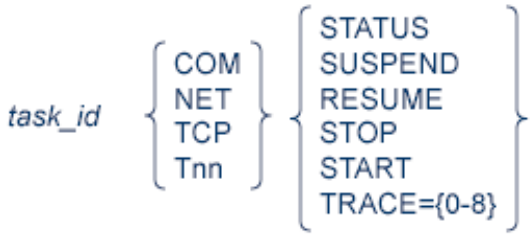
- Enter command:

```
task_id SECURITY TRACE=2
```

See `TRACE-LEVEL` under *Broker Attributes* in the platform-independent administration documentation.

Transport-specific Commands

Transport-specific commands are available for Adabas/Entire Net-Work communicators and TCP communicators; the COM command can be used for all communicators. The following command syntax applies:



COM parameter

This command is executed by all configured transport communicators. The following parameters are supported:

Parameter	Description
STATUS	Displays the current status of the transport communicator.
SUSPEND	Used to suspend the transport communicator. The transport communicator is halted but will not shut down. User requests receive response code 148.
RESUME	Resume a suspended transport communicator. If the communicator was not suspended before, an error message will be displayed.
STOP	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.
START	Start a transport communicator that was previously stopped. If the communicator was not stopped before, an error message will be displayed.
TRACE	Sets the trace level for the transport method. If the global trace level (see TRACE) is set with command *task_id* TRACE=*n* this applies to *all* transport methods. This command will also override any existing transport-specific settings. If you subsequently enter command *task_id* TCP TRACE=*n* only the trace level for TCP/IP transport is modified. **Note:** With command TCP T*nn*, the trace level is set for *all* TCP communicators. Setting a trace level for a single TCP instance is not supported.

Sample Output

```
ETBM0720 Operator typed in: COM STATUS
ETBW0718 TCP Communicator 0 currently active
ETBW0718 TCP Communicator 1 currently active
ETBW0718 NET Communicator 0 currently suspended
XCO0039I 00113 Total number of commands = 17
XCO0057I 00113 Operator entry active
ETBM0720 Operator typed in: COM SUSPEND
ETBM0721 TCP Communicator 0 suspended
ETBM0721 TCP Communicator 1 suspended
ETBM0721 NET Communicator 0 suspended
```

NET parameter

This command is executed by X-COM, the Adabas/Entire Net-Work communicator. See command `COM` above for a list of supported parameters.

Sample Output

```
ETBM0720 Operator typed in: NET STATUS
ETBW0718 NET Communicator 0 currently active
XCO0039I 00113 Total number of commands = 17
XCO0057I 00113 Operator entry active
```

TCP parameter

This command is executed by TCP communicators. See command `COM` above for a list of supported parameters.

Sample Output

```
ETBM0720 Operator typed in: TCP STATUS
ETBW0718 TCP Communicator 0 currently active
ETBW0718 TCP Communicator 1 currently active
```

```
ETBM0720 Operator typed in: TCP RESUME
ETBM0721 TCP Communicator 0 resumed
ETBM0721 TCP Communicator 1 resumed
```

To manipulate a specific communicator instance (max. five instances can be started), use the command `T00`, `T01`, `T02`, `T03` or `T04` for the respective TCP instance.

Sample Output

```
ETBM0720 Operator typed in: T00 STATUS
ETBW0718 TCP Communicator 0 currently active
```

```
ETBM0720 Operator typed in: T01 STATUS
ETBW0718 TCP Communicator 1 currently active
```

Sample Transport Commands

To display status of all transport communicators

- Enter command:

```
task_id COM STATUS
```

To suspend first TCP communicator

- Enter command:

```
task_id T00 SUSPEND
```

XCOM-specific Commands

Note: All operator commands beginning with "X" belong to X-COM, the Adabas/Entire Net-Work communicator. The following commands operate only on the Adabas transport mechanism: `XCQES`, `XHALT`, `XPARM`, `XSTART`, `XSTAT` and `XUSER`. These commands have no effect on functions not related to the Adabas transport mechanism.

`XEND` and `XSTOP` function independently of the transport mechanism. (They stop the Broker's processing immediately, whereby existing calls to the EntireX Broker are not allowed to finish.)

XABS

Displays the current number, and the highest number, of used bytes in the Adabas attached buffer pool to the console.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

Sample Output

```
ETBM0720 Operator typed in: XABS
XC00090I 00113 Attached buffer usage
XC00091I 00113 Number of bytes in use = 0
XC00092I 00113 Highest number of bytes in use = 6400
```

XCQES

Displays the current number, and the highest number, of Adabas command queue elements to the console.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

Sample Output

```
ETBM0720 Operator typed in: XCQES
XC00030I 00113 Number of active CQEs = 0
XC00031I 00113 Highest number of active CQEs = 1
```

XEND

Alias of ETBEND.

XHALT

New calls to the EntireX Broker are temporarily rejected. Processing is resumed by issuing the XSTART operator command. XHALT is an alias for command NET SUSPEND.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

Sample Output

```
ETBM0720 Operator typed in: XHALT
ETBM0721 NET Communicator 0 suspended
```

XPARM

Displays the values of Adabas SVC, database ID, number of CQEs, number of attached buffers, and the application name for the Adabas transport to the console.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

Sample Output

```
ETBM0720 Operator typed in: XPARM
XC00032I 00113 Parameters for this session:
XC00033I 00113 SVC = 249
XC00034I 00113 NODE = 00113
XC00035I 00113 NCQE = 00100
XC00036I 00113 NABS = 10000
XC00037I 00113 User application = ETBNUC
```

XSTART

Processing of new calls to the EntireX Broker, interrupted with the XHALT command, is resumed. XSTART is an alias of command NET RESUME.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

Sample Output

```
ETBM0720 Operator typed in: XSTART
ETBM0721 NET Communicator 0 resumed
```

XSTAT

Displays the EntireX Broker statistics as console messages.

Note: This command operates on the Adabas transport mechanism only. It has no effect on functions not related to the Adabas transport mechanism.

XSTOP

Alias of `ETBEND`.

XUSER

Displays the current number, as well as the highest number, of users actively issuing commands using the Adabas transport mechanism to the console.

Note: The number of users displayed with this operator command will not represent all of the Broker clients and servers but only the subset of users issuing commands using the Adabas transport mechanism. Command and Information Services provides comprehensive information about all Broker clients and servers.

4 Broker Command-line Utilities

EntireX Broker provides the following internal services: Command Service and Information Service, which can be used to administer and monitor brokers. Because these services are implemented internally, nothing has to be started or configured. You can use these services immediately after starting EntireX Broker.

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. `ETBINFO` uses data descriptions called profiles to control the type of data that is returned for a request. `ETBINFO` is useful for monitoring and administering EntireX Broker efficiently, for example how many users can run concurrently and whether the number of specified message containers is large enough.

Although basic formatting of the output is available, it is usually formatted by script languages or other means external to the Broker.

- Running the Command-line Utility
- Command-line Parameters
- Profile
- Format String

Running the Command-line Utility

In a z/VSE environment, run the command-line utility `ETBINFO` as shown below:

```
* $$ JOB JNM=RUNINFO,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=H
// JOB RUNINFO
*
* INFORMATION SERVICES SAMPLE JCL
*
// LIBDEF *,SEARCH=(SAGLIB.EXX960,SAGLIB.WAL826)
/*
/* / EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-d BROKER -c PING -bip:+
/*               port:TCP'
// EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-d BROKER -c PING -bETBn+
               nnnn:SVCmmm:NET'
/*
// EXEC LISTLOG
/&
* $$ EOJ
```

Command-line Parameters

The table below explains the command-line parameters. The format string and profile parameters are described in detail following the table. All entries in the Option column are case-sensitive.

Option	Command-line Parameter	Req/ Opt	Explanation
-b	brokerid	R	Broker identifier, for example localhost:1971:TCP.
-c	class	O	Class as selection criterion.
-C	csvoutput	O	Comma-separated values, suitable for input into a spreadsheet or other analysis tool. Any format string specified by means of format string or profile command-line parameters is ignored.
-d	object	R	Possible values: Object — Provides Info on BROKER — Broker. CLIENT — Client. CMDLOG-FILTER — Command log filter. CONVERSATION — Conversation. NET — Entire Net-Work transport. PARTICIPANT — Participant. POOL-USAGE — Broker pool usage. PSF — Unit-of-work status. PSFADA — Adabas persistent store. PSFCTREE — c-tree persistent store. PSFDIV — DIV persistent store. PSFFILE — FILE persistent store. PUBLICATION — Publication. PUBLISHER — Publisher. RESOURCE-USAGE — Broker resource usage. SECURITY — EntireX Security. SERVER — Server. SERVICE — Service. STATISTICS — Broker statistics. SUBSCRIBER — Subscriber. TCP — TCP transport. TOPIC — Topic. USER — Participant (short).

Option	Command-line Parameter	Req/ Opt	Explanation
			WORKER Worker. WORKER-USAGE Worker usage.
-e	recv class	O	Receiver's class name. This selection criterion is valid only for object PSF.
-f	*Format String*	O	Format string how you expect the output. See *Profile*.
-g	recv service	O	Receiver's service name. This selection criterion is valid only for object PSF.
-h	help	O	Prints help information.
-i	convid	O	Conversation ID as selection criterion. Only valid for object CONVERSATION.
-I	conv type	O	Conversation's type.
-j	recv server	O	Receiver's server name. This selection criterion is valid only for object PSF.
-k	recv token	O	Receiver's token. This selection criterion is valid only for object PSF.
-l	level	O	The amount of information displayed: FULL All information. SHORT User-specific information.
-m	recv userid	O	Receiver's user ID. This selection criterion is valid only for object PSF.
-n	server name	O	Server name. This selection criterion is valid only for the objects SERVER, SERVICE or CONVERSATION.
-p	library.sublibrary(profile.pro)	O	Here you can specify a sublibrary element that defines the layout of the output. There are default files you can modify or you can use your own. The default files are: BROKER CLIENT CLOGFLT CONV NET POOL PSF PSFADA PSFCTREE PSFDIV PSFFILE PUBLIC PUBSHR RESOURCE SECURITY SERVER SERVICE STATIS SUBSCBR TCP TOPIC USER WORKER WKRUSAGE See *Profile*.
-q	puserid	O	Physical user ID. This selection criterion is valid only for objects CLIENT, SERVER, CONVERSATION, SUBSCRIBER, PUBLISHER or PUBLICATION. **Note:** Must be a hex value.

Option	Command-line Parameter	Req/ Opt	Explanation
-P	publication id	O	Publication ID. This selection criterion is valid only for object PUBLICATION.
-r	sec	O	Refresh information after seconds.
-s	service	O	Service. This selection criterion is valid only for objects SERVER, SERVICE or CONVERSATION.
-t	token	O	This selection criterion is valid only for objects CLIENT, SERVER, SERVICE, CONVERSATION, SUBSCRIBER, PUBLISHER, PUBLICATION or TOPIC.
-T	topic	O	Topic name. This selection criterion is valid only for objects PUBLICATION, SUBSCRIBER, PUBLISHER, or TOPIC.
-u	userid	O	User ID. This selection criterion is only valid for the display types CLIENT, SERVER, SERVICE, CONVERSATION, SUBSCRIBER, PUBLISHER, PUBLICATION or TOPIC.
-U	subscr type	O	Subscriber's subscription type. This selection criterion is valid only for object SUBSCRIBER.
-v	UOW status	O	Unit of work status. This selection criterion is valid only for object PSF.
-w	UOW ID	O	Unit of work ID. This selection criterion is valid only for object PSF.
-x	userid	O	User ID. For security purposes.
-y	password	O	Password. For security purposes.
-z	token	O	Used with userid to uniquely identify a caller to Command and Information Services.

Profile

If you do not use the profile option or a format string, your output will be an unformatted list with all columns of that display type. To display specific columns, specify a profile that includes only those columns.

The following default sample profiles include all the columns defined for each display type:

- BROKER
- CLIENT
- CLOGFLT
- CONV
- POOL
- PSF
- PSFCTREE
- PSFDIV
- PSFFILE
- PUBLIC
- PUBSHR
- RESOURCE
- SERVICE
- STATIS
- SUBSCBR
- TCP
- TOPIC
- USER

■ PSFADA ■ SECURITY ■ WKRUSAGE
■ SERVER ■ WORKER

You can either delete the columns not required or copy the default profile and modify the order of the columns. Ensure that the column names have a leading "%". Column names can be written in one line or on separate lines. The output is always written side by side.

Location of Profiles

On z/VSE, the profiles used to control the format of the data displayed are members of the EXX960 sublibrary and are named `SERVER.PRO`, `CLIENT.PRO` etc.

Example for using a profile:

```
// EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-b LOCALHOST:1971:TCP    +
               -d SERVICE -p DD:SAGLIB.EXX960(SERVICE.PRO)'
```

Format String

The format string, if specified, will override the use of a profile. The format string is built like a `printf()` in C language. The string must be enclosed in quotation marks. You can specify the columns by using a "%" and the column name. The column name must contain letters only. Numeric characters are not allowed. You can specify the length of column output by using a format precision, as in the ANSI-C `printf()` function. The column name must be followed by a blank. For example:

```
// EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-b LOCALHOST:1971:TCP    +
-d SERVICE',                                              +
PARM=' -f "CLASS: %24SERVER-CLASS   SERVER: %24SERVER-NA+
ME   SERVICE: %24SERVICE"'
```

which produces:

```
CLASS:                      SAG   SERVER:                  ETBCIS   SERVICE:     ↩
                INFO
CLASS:                      SAG   SERVER:                  ETBCIS   SERVICE:     ↩
          USER-INFO
CLASS:                      SAG   SERVER:                  ETBCIS   SERVICE:     ↩
                 CMD
CLASS:                      SAG   SERVER:                  ETBCIS   SERVICE:     ↩
PARTICIPANT-SHUTDOWN
CLASS:                      SAG   SERVER:                  ETBCIS   SERVICE:     ↩
        SECURITY-CMD
```

Example:

```
// EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-b DAVLCSI:9084:TCP      +
               -d BROKER',                                             +
               PARM=' -f "%12.12CPLATNAME   %NUM-SERVER   %NUM-CLIENT"'
```

which produces:

```
z/VSE 5.1.2   12   200
```

You can also use an arbitrary column separator, which can be any character other than "%". You can use `\n` for a new line in the output and `\t` for a tabulator in the format string or profile. Please note that due to the `PARM` string syntax in the z/VSE `EXEC` command, `\n` becomes `\\n`. For example:

```
// EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-b LOCALHOST:1971:TCP    +
               -d SERVICE',                                            +
               PARM=' -f "CLASS: %SERVER-CLASS   \\n\\tSERVER: %SERVER-+
               NAME   \\n\\tSERVICE: %SERVICE"'
```

which produces:

```
CLASS: SAG
      SERVER: ETBCIS
      SERVICE: INFO
CLASS: SAG
      SERVER: ETBCIS
      SERVICE: USER-INFO
CLASS: SAG
      SERVER: ETBCIS
      SERVICE: CMD
CLASS: SAG
      SERVER: ETBCIS
      SERVICE: PARTICIPANT-SHUTDOWN
CLASS: SAG
      SERVER: ETBCIS
      SERVICE: SECURITY-CMD
```

ETBCMD

Allows the user to take actions - for example purge a unit of work, stop a server, shut down a Broker - against EntireX Broker.

Running the Command-line Utility

In a z/VSE environment, run the `ETBCMD` command-line utility like this:

```
* $$ JOB JNM=RUNCMD,CLASS=0,DISP=D
* $$ LST CLASS=A,DISP=H
// JOB RUNCMD
*
* COMMAND SERVICES SAMPLE JCL
*
// LIBDEF *,SEARCH=(SAGLIB.EXX960,SAGLIB.WAL826)
/*
/* / EXEC ETBCMD,PARM='ENVAR("LOGNAME=ENTIRE")/-d BROKER -c PING -bip:+
/*               port:TCP'
// EXEC ETBCMD,PARM='ENVAR("LOGNAME=ENTIRE")/-d BROKER -c PING -bETBnn+
              nnn:SVCmmm:NET'
/*
// EXEC LISTLOG
/amp;
* $$ EOJ
```

Command-line Parameters

The table below explains the command-line parameters. All entries in the **Option** column are case-sensitive.

Command-line Parameter	Option	Parameter	Req/ Opt	Explanation
`brokerid`	`-b`	e.g. `ETB001`	R	Broker ID.
`command`	`-c`	■ `ALLOW-NEWUOWMSGS` ■ `CLEAR-CMDLOG-FILTER` ■ `CONNECT-PSTORE` ■ `DISABLE-ACCOUNTING` ■ `DISABLE-CMDLOG-FILTER`	R	Command to be performed. See *List of Commands and Objects* below.

Command-line Parameter	Option	Parameter	Req/ Opt	Explanation
		■ DISABLE-CMDLOG ■ DISABLE-DYN-WORKER ■ DISCONNECT-PSTORE ■ ENABLE-ACCOUNTING ■ ENABLE-CMDLOG-FILTER ■ ENABLE-CMDLOG ■ ENABLE-DYN-WORKER ■ FORBID-NEWUOWMSGS ■ PING ■ PRODUCE-STATISTICS ■ PURGE ■ RESET-USER ■ RESUME ■ SET-CMDLOG-FILTER ■ SHUTDOWN ■ START ■ STATUS ■ STOP ■ SUBSCRIBE ■ SUSPEND ■ SWITCH-CMDLOG ■ TRACE-FLUSH ■ TRACE-OFF ■ TRACE-ON ■ TRAP-ERROR ■ UNSUBSCRIBE		
object type	-d	■ BROKER ■ CONVERSATION ■ PARTICIPANT ■ PSF ■ SUBSCRIBER ■ SECURITY ■ SERVER	R	The object type to be operated on. See ***List of Commands and Objects*** below. Within EntireX Broker nomenclature, a participant is an application implicitly or explicitly logged on to the Broker as a specific user. A participant could act as client, server, publisher or subscriber.

Command-line Parameter	Option	Parameter	Req/ Opt	Explanation
		■ SERVICE ■ TRANSPORT		
	-e	errornumber	O	Error number being trapped.
	-E		O	Exclude attach servers from service shutdown.
help	-h		O	Prints help information.
class/server/service	-n	class/server/service	O	Service triplet.
option	-o	■ IMMED ■ QUIESCE ■ LEVEL*n*, where *n*=1-8	O	Command option.
puserid	-p	puserid	O	Physical User ID. For SERVER and PARTICIPANT objects only. This must be a hex value.
seqno	-S	sequence number	O	Sequence number of participant.
token	-t	token	O	Token. For PARTICIPANT and SUBSCRIBER objects only.
topic	-T	topic	O	Topic name. For SUBSCRIBER object only.
uowid	-u	uowid	O	Unit of work ID. For PSF object only.
userid	-U	userid	O	User ID. For PARTICIPANT and SUBSCRIBER objects only.
secuserid	-x	userid	O	User ID for security purposes.
transportid	-X	Transport ID	O	One of the following: COM|NET|TCP|T*nn*. See table below.
secpassword	-y	password	O	Password for security purposes.

Transport ID Values

This table explains the possible values for parameter transportid:

Transport ID	Explanation
COM	all communicators
NET	NET transport communicator
TCP	all TCP/IP communicators
T00	TCP/IP communicator 1
T01	TCP/IP communicator 2
T02	TCP/IP communicator 3

Transport ID	Explanation
T03	TCP/IP communicator 4
T04	TCP/IP communicator 5

List of Commands and Objects

This table lists the available commands and the objects to which they can be applied.

Command	Object								
	BROKER	CONVERSATION	PARTICIPANT	PSF	SECURITY	SERVER	SERVICE	SUBSCRIBER	TRANSPORT
ALLOW-NEWUOWMSGS				x					
CLEAR-CMDLOG-FILTER	x								
CONNECT-PSTORE				x					
DISABLE-ACCOUNTING	x								
DISABLE-CMDLOG-FILTER	x								
DISABLE-CMDLOG	x								
DISCONNECT-PSTORE				x					
ENABLE-ACCOUNTING	x								
ENABLE-CMDLOG-FILTER	x								
ENABLE-CMDLOG	x								
FORBID-NEWUOWMSGS				x					
PING	x								
PRODUCE-STATISTICS	x								
PURGE				x					
RESET-USER					x				
SET-CMDLOG-FILTER	x								
SHUTDOWN	x	x	x			x	x		
START									x
STATUS									x
STOP									x
SUBSCRIBE								x	
SWITCH-CMDLOG	x								
TRACE-OFF	x			x	x				
TRACE-ON	x			x	x				
UNSUBSCRIBE								x	

Examples

ETBCMD **Example** PARM **Strings**	**Description**
PARM='ENVAR("LOGNAME=ENTIRE")/-h'	Displays ETBCMD help text.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d BROKER -c TRACE-OFF'	Turns Broker tracing off.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d BROKER -c TRACE-ON -o LEVEL2'	Sets Broker trace level to 2.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d BROKER -c SHUTDOWN'	Performs Broker shutdown.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d SERVICE -c SHUTDOWN -o IMMED -n ACLASS/ASERVER/ASERVICE'	Shutdown service CLASS=ACLASS,SERVER=ASERVER,SERVICE=ASERVICE. See also *SHUTDOWN SERVICE* under *Broker Command and Information Services* for more information on shutdown options.
	Create list of servers and shutdown specific server in two steps (first step uses ETBINFO). See also *SHUTDOWN SERVER* under *Broker Command and Information Services*.
EXEC ETBINFO,PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d SERVER -l FULL -f"%USER-ID %SEQNO"'	1. Determine a list of all servers with sequence numbers.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d SERVER -c SHUTDOWN -o IMMED -S32'	2. Shutdown server with sequence number 32.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d BROKER -c PING'	Performs an EntireX ping against the Broker.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d PSF -c DISCONNECT-PSTORE'	Disconnects the Broker PSTORE.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d PSF -c CONNECT-PSTORE'	Connects the Broker PSTORE.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d PSF -c PURGE -u 100000000U00001A'	Purges a unit of work.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d PSF -c ALLOW-NEWUOWMSGS'	Allows new units of work to be stored.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d PSF -c FORBID-NEWUOWMSGS'	Disallows new units of work to be stored.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d SUBSCRIBER -c SUBSCRIBE -U U1 -t t1 -T NYSE'	Subscribes subscriber to topic NYSE.
PARM='ENVAR("LOGNAME=ENTIRE")/-b etb001 -d SUBSCRIBER -c UNSUBSCRIBE -U U1 -t t1 -T NYSE'	Unsubscribes subscriber from topic NYSE.

5 Configuring Broker for Internationalization

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

This chapter explains in detail how to configure the broker for the various internationalization approaches, how to write a translation user exit and how to write a SAGTRPC user exit.

See also *What is the Best Internationalization Approach to use?* under *Introduction to Internationalization*

Configuring Translation

To configure translation

- In the Broker attribute file, set the service-specific or topic-specific broker attribute `TRANSLATION` to `SAGTCHA` as the name of the translation routine. Example:

```
TRANSLATION=SAGTCHA
```

Configuring Translation User Exits

To configure translation user exits

As a prerequisite, the user-written translation module must be accessible to the Broker worker threads.

1 Copy the user-written translation module into any sublibrary defined to EntireX Broker's `LIBDEF` chain.

2 In the Broker attribute file, set the service-specific or topic-specific broker attribute `TRANSLATION` to the name of the user-written translation routine. Example:

```
TRANSLATION=MYTRANS
```

Configuring ICU Conversion

To configure ICU conversion

1 In the Broker attribute file, set the `ICU-CONVERSION` (default for z/VSE is `NO`):

```
ICU-CONVERSION=YES
```

2 In the Broker attribute file, set the service-specific or topic-specific broker attribute `CONVERSION`. Default for z/VSE is `NO`. Examples:

- ICU Conversion with SAGTCHA for *ACI-based Programming*:

```
CONVERSION=(SAGTCHA,TRACE=1,OPTION=SUBSTITUTE)
```

- ICU Conversion with SAGTRPC for *RPC-based Components* and *Reliable RPC*:

```
CONVERSION=(SAGTRPC,TRACE=2,OPTION=STOP)
```

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*.

3 Optionally configure a `CONVERSION OPTION` to tune error behavior to meet your requirements; see *`OPTION` Values for Conversion*.

To configure locale string defaults (optional)

- If the broker's locale string defaults do not match your requirements (see *Broker's Locale String Defaults* under *Locale String Mapping* in the internationalization documentation), we recommend you assign suitable locale string defaults for your country and region, see the respective attribute in *Codepage-specific Attributes* (`DEFAULTS=CODEPAGE`) under *Broker Attributes* in the platform-independent administration documentation for how to customize the broker's locale string defaults.

To customize mapping of locale strings (optional)

- If the built-in locale string mapping mechanism does not match your requirements, you can assign specific codepages to locale strings. See *Broker's Built-in Locale String Mapping* under *Locale String Mapping* in the internationalization documentation and `locale-string` for information on customizing the mapping of locale strings to codepages.

Writing Translation User Exits

This section covers the following topics:

Introduction

EntireX Broker provides an interface to enable user-written translation routines in the programming language . It contains three parameters:

- The address of the TRAP control block (TRAP = Translation Routine / Area for Parameters).
- The address of a temporary work area. It is aligned to fullword / long integer boundary (divisible by 4). The work area can only be used for temporary needs and is cleared after return.
- A fullword (long integer) that contains the length of the work area.

Note: Names for user-written translation routines starting with "SAG" are reserved for Software AG usage and must not be used, e.g. "SAGTCHA" and "SAGTRPC".

Structure of the TRAP Control Block

The Assembler dummy section TR$TRAP covers the layout of the TRAP control block:

```
TR$TRAP  DSECT ,
TR$TYPE  DS       F            TRAP type
TR$TYP2  EQU      2            TRAP type ETB 121
TR$ILEN  DS       F            Input buffer length
TR$IBUF  DS       A            Address of input buffer
TR$OLEN  DS       F            Output buffer length
TR$OBUF  DS       A            Address of output buffer
TR$DLEN  DS       F            Length of data returned:
*                              Should be set to the minimum value of TR$ILEN
*                              and TR$OLEN.
TR$SHOST DS       F            Sender's host:
*                              x'00000000' = little endian
*                              x'00000001' = big endian
TR$SCODE DS       F            Sender's character set:
SEBCIBM  EQU      X'00000022'  EBCDIC (IBM)
SEBCSNI  EQU      X'00000042'  EBCDIC (SNI)
SA88591  EQU      X'00000080'  ASCII
TR$RHOST DS       F            Receiver's host     --> see TR$SHOST
TR$RCODE DS       F            Receiver's char set --> see TR$SCODE
TR$BHOST DS       F            BROKER host         --> see TR$SHOST
TR$BCODE DS       F            BROKER char set     --> see TR$SCODE
TR$SENVA DS       F            Sender's ENVIRONMENT field supplied:
OFF      EQU      X'00000000'  ENVIRONMENT field not set
ON       EQU      X'00000001'  ENVIRONMENT field set
*
TR$RENVA DS       F            Receiver's ENVIRONMENT field supplied:
*                              --> see TR$SENVA
TR$SENV  DS       CL32         Sender's ENVIRONMENT field
TR$RENV  DS       CL32         Receiver's ENVIRONMENT field
TR$LEN   EQU      *-TR$TRAP    Length of TRAP
```

The translation routine USRTCHA is an example of the translation user exit, it is contained in the EntireX Common source library.

Using the TRAP Fields

The TR$DLEN must be supplied by the user-written translation routine. It tells the Broker the length of the message of the translation. In our example its value is set to the minimum length of the input and output buffer.

All other TRAP fields are supplied by the Broker and must not be modified by the user-written translation routine.

The incoming message is located in a buffer pointed to by TR$IBUF. The length (not to be exceeded) is supplied in TR$ILEN. The character set information from the send buffer can be taken from TR$SCODE.

The outgoing message must be written to the buffer pointed to by TR$OBUF. The length of the output buffer is given in the field TR$OLEN. The character set is specified in TR$RCODE. If the addresses given in TR$IBUF and TR$OBUF point to the same location, it is not necessary to copy the data from the input buffer to the output buffer.

The environment fields TR$SENVA and TR$RENVA are provided to handle site-dependent character set information. For the SEND and/or RECEIVE functions, you can specify data in the ENVIRONMENT field of the Broker ACI control block. This data is translated into the codepage of the platform where EntireX Broker is running (see field TR$BCODE) and is available to the TR$SENV or TR$RENV field in the TRAP control block. TR$SENVA or TR$RENVA are set to ON if environmental data is available.

The sample source USRTCHA contains a section to handle the ENVIRONMENT value *NONE. The translation will be skipped if *NONE is supplied by the sender or receiver. Any values given in the API field ENVIRONMENT must correspond to the values handled in the translation routine.

6 Managing the Broker Persistent Store

The persistent store is used for storing unit-of-work messages and publish-and-subscribe data to disk. This means message and status information can be recovered after a hardware or software failure to the previous commit point issued by each application component. Under z/VSE, the broker persistent store can be implemented with the Adabas database of Software AG. This chapter covers the following topics:

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

Implementing an Adabas Database as Persistent Store

Introduction

EntireX provides an Adabas persistent driver. This enables Broker unit of work (UOW) messages and their status to be stored in an Adabas file. It is designed to work with Adabas databases under z/OS, UNIX, Windows, BS200/OSD and z/VSE, and can be used where the database resides on a different machine to Broker kernel. For performance reasons, we recommend using EntireX Broker on the same machine as the Adabas database.

Configuring and Operating the Adabas Persistent Store

Selecting the Adabas Persistent Store Driver

To create an Adabas Persistent Store, adapt and run job `PSFADA.J` in sublibrary `EXX960`. Running this job will load an empty Adabas persistent store file into your Adabas database. To activate the Adabas persistent store, set up at least following parameters in the broker attribute file. See *Broker Attributes* in the platform-independent administration documentation.

```
DEFAULTS=BROKER
  STORE         = BROKER
  PSTORE        = HOT/COLD
  PSTORE-TYPE   = ADABAS
DEFAULTS=ADABAS
  DBID          = dbid
  FNR           = pstore_file_number
DEFAULTS=NET
  ADASVC        = svc_number
```

See *Managing the Broker Persistent Store* for more information.

Restrictions

If a HOT start is performed, the Broker kernel must be executed on the same platform on which also the previous Broker executed. This is because some portions of the persistent data are stored in the native character set and format of the Broker kernel. It is also necessary to start Broker with the same Broker ID as the previous Broker executed.

If a COLD start is executed, a check is made to ensure the Broker ID and platform information found in the persistent store file is consistent with the Broker being started (provided the persistent store file is not empty). This is done to prevent accidental deletion of data in the persistent store by a different Broker ID. If you intend to COLD start Broker and to utilize a persistent store file which has been used previously by a different Broker ID, you must supply the additional `PSTORE-TYPE` parameter `FORCE-COLD=Y`.

Recommendations

- Perform regular backup operations on your Adabas database. The persistent store driver writes C1 checkpoint records at each start up and shut down of Broker.
- For performance reasons, execute Broker on the same machine as Adabas.

Broker Checkpoints in Adabas

During startup, Broker writes the following C1 checkpoint records to the Adabas database. The time, date and job name are recorded in the Adabas checkpoint log. This enables Adabas protection logs to be coordinated with Broker executions. This information can be read from Adabas, using the `ADAREP` utility with option `CPLIST`:

Broker Execution Name	Broker Execution Type	Adabas
ETBC	Broker Cold Start	Normal Cold Start
ETBH	Broker Hot Start	Normal Hot Start
ETBT	Broker Termination	Normal Termination

Adabas DBA Considerations

BLKSIZE : Adabas Persistent Store Parameter for Broker

Caution should be exercised when defining the block size (`BLKSIZE`) parameter for the Adabas persistent store. This determines how much UOW message data can be stored within a single Adabas record. Therefore, do not define a much larger block size than the size of the maximum unit of work being processed by Broker. (Remember to add 41 bytes for each message in the unit of work.) The advantage of having a good fit between the unit of work and the block size is that fewer records are required for each I/O operation.

It is necessary to consider the following Adabas parameters and settings when using Adabas for the persistent store file:

Table of Adabas Parameter Settings

Topic	Description
Allowing Sufficient Adabas UQ Elements	Allow sufficient Adabas user queue (UQ) elements each time you start Broker. The Broker utilizes a number of user queue elements equal to the number of worker tasks (`NUM-WORKER`), plus two. Adabas timeout parameter (`TNAE`) determines how long the user queue elements will remain. This can be important if Broker is restarted after an abnormal termination, and provision must be made for sufficient user queue elements in the event of restarting Broker.
Setting Size of Hold Queue Parameters	Consideration must be given to the Adabas hold queue parameters `NISNHQ` and `NH`. These must be sufficiently large to allow Adabas to add/update/delete the actual number of records within a single unit of work. Example: where there are 100 message within a unit of work and the average message size is 10,000 bytes, the total unit of work size is 1 MB. If, for example, a 2 KB block size (default `BLKSIZE=2000`) is utilized by the Adabas persistent store driver, there will be 500 distinct records within a single Adabas commit (`ET`) operation, and provision must be made for this to occur successfully.
Setting Adabas TT Parameter	Consideration must be given to the Adabas transaction time (`TT`) parameter for cases where a large number of records is being updated within a single unit of work.
Defining LWP Size	Sufficient logical work pool (`LWP`) size must be defined so that the Adabas persistent store can update and commit the units of work. Adabas must be able to accommodate this in addition to any other processing for which it is used.

Topic	Description
Executing Broker Kernel and Adabas Nucleus on Separate Machines	If Broker kernel is executed on a separate machine to the Adabas nucleus, with a different architecture and codepage, then we recommend running the Adabas nucleus with the `UEC` (universal conversion) option in order to ensure that Adabas C1 checkpoints are legible within the Adabas checkpoint log.
Setting `INDEXCOMPRESSION=YES`	This Adabas option can be applied to the Adabas file to reduce by approximately 50% the amount of space consumed in the indexes.
4-byte ISNs	If you anticipate having more than 16 million records within the persistent store file, you must use 4-byte ISNs when defining the Adabas file for EntireX.
Specification of Adabas LP Parameter	**Caution:** This parameter must be specified large enough to allow the largest UOW to be stored in Adabas. If this is not large enough, Broker will detect an error (response 9; subresponse - 4 bytes - X'0003',C'LP') and Broker will not be able to write any further UOWs. See the description of the `LP` parameter under *ADARUN Parameters* in the*DBA Reference Summary* of the Adabas documentation.

Estimating the Number of Records to be Stored

To calculate the Adabas file size it is necessary to estimate the number of records being stored. As an approximate guide, there will be one Adabas record (500 bytes) for each unprocessed unit of work, plus also *n* records containing the actual message data, which depends on the logical block size and the size of the unit of work. In addition, there will be one single record (500 bytes) for each unit of work having a persisted status.

Always allow ample space for the Adabas persistent store file since the continuous operation of Broker relies of the availability of this file to store and retrieve information.

Estimating the Number of Records to be Stored

In this example there are 100,000 Active UOW records at any one time. Each of these is associated with two message records containing the message data. UOW records are 500 bytes in length. Each message record contains 2,000 bytes. In addition, there are 500,000 UOW status records residing in the persistent store, for which the UOW has already been completely processed. These are 500 bytes long.

Note: The actual size of the data stored within the UOW message records is the sum of all the messages within the UOW, plus a 41-byte header for each message. Therefore, if the average message length is 59 bytes, the two 2,000 bytes, messagesrecords, could contain n = 4,000 / (59+41), or 40 messages. Adabas is assumed to compress the message data by 50% in the example (this can vary according to the nature of the message data).

3-byte ISNs and RABNs are assumed in this example. A device type of 8393 is used; therefore, the ASSO block size is 4,096, and DATA block size is 27,644. Padding factor of 10% is specified.

The following example calculates the space needed for Normal Index (NI), Upper Index (UI), Address Converter (AC) and Data Storage (DS).

Calculation Factors	Required Space
■ Number entries for descriptor WK (21-byte unique key)	■ = number UOW records: 0.1 + 0.5 million + number message records: 0.2 million
■ NI Space for descriptor WK ■ (3-byte ISN) ■ (4,092 ASSO block 10% padding)	■ = 800,000 * (3 + 21 + 2) ■ = 20,800,000 bytes ■ = 5,648 blocks
■ UI Space for descriptor WK ■ (3-byte ISN + 3-byte RABN) ■ (4,092 ASSO block 10% padding)	■ = 5,648 * (21 + 3 + 3 + 1) ■ = 158,140 bytes ■ = 43 blocks
■ Number entries for descriptor WI (8-byte unique key)	■ = number processed UOW records: 0.5 million
■ NI Space for descriptor WI ■ (3-byte ISN) ■ (4,092 ASSO block 10% padding)	■ = 500,000 * (3 + 8 + 2) ■ = 6,500,000 bytes ■ = 1,765 blocks
■ UI Space for descriptor WI ■ (3-byte ISN and 3 byte RABN) ■ (4,092 ASSO block 10% padding)	■ = 17,649 * (8 + 3 + 3 + 1) ■ = 26,475 bytes ■ = 8 blocks
■ Number entries for descriptor WL (96 byte key)	■ = number UOW records 0.1 + 0.5 million
■ NI Space for descriptor WL ■ (3-byte ISN) ■ (4,092 ASSO block 10% padding)	■ = 600,000 * (3 + 96 + 2) ■ = 60,600,000 bytes ■ = 16,455 blocks
■ UI Space for descriptor WL ■ (3-byte ISN and 3 byte RABN) ■ (4,092 ASSO block 10% padding)	■ = 164,548 * (96 + 3 + 3 + 1) ■ = 16,948,517 bytes ■ = 461 blocks
■ Address Converter space ■ (4,092 ASSO block)	■ = (800,000 + 1) * 3 / 4092 ■ = 587 blocks

Calculation Factors	Required Space
■ Data storage for message data (2,000-byte records compressed by 50%)	■ = 0.2 million * 2000 * 0.5 = 200,000,000 bytes
■ Data storage for UOW data (2,000-byte records compressed by 50%)	■ = 0.6 million * 500 * 0.5 = 150,000,000 byte
■ Combined space required for data (27,644 DATA block 10% padding)	■ = 14,068 blocks
Entity Requiring Space	**Total Required Space**
Normal Index (NI)	= 23,868 blocks
Upper Index (UI)	= 512 blocks
Data Storage (DS)	= 14,068 blocks
Address Converter (AC)	= 587 blocks

Tips on Transports, Platforms and Versions

- **Entire Net-Work**
  If you intend to use Adabas persistent store through Entire Net-Work, see the Entire Net-Work documentation for installation and configuration details.
- **Adabas Versions**
  Adabas persistent store can be used on all Adabas versions currently released and supported by Software AG.
- **Prerequisite Versions of Entire Net-Work with Adabas**
  See the Adabas and Entire Net-Work documentation to determine prerequisite versions of Entire Net-Work to use with Adabas at your site.

Migrating the Persistent Store

The contents of EntireX Broker's persistent store can be migrated to a new persistent store in order to change the PSTORE type or to use the same type of PSTORE with increased capacity.

The migration procedure outlined here requires two Broker instances started with a special `RUN-MODE` parameter. One Broker unloads the contents of the persistent store and transmits the data to the other Broker, which loads data into the new PSTORE. Therefore, for the purposes of this discussion, we will refer to an *unload* Broker and a *load* Broker.

This procedure is based on Broker-to-Broker communication to establish a communication link between two Broker instances. It does not use any conversion facilities, since the migration procedure is supported for homogeneous platforms only.

Configuration

The migration procedure requires two Broker instances started with the `RUN-MODE` parameter. The unload Broker should be started with the following attribute:

```
RUN-MODE=PSTORE-UNLOAD
```

The load Broker should be started with the following attribute:

```
RUN-MODE=PSTORE-LOAD
```

These commands instruct the Broker instances to perform the PSTORE migration.

Note: The attribute `PARTNER-CLUSTER-ADDRESS` must be defined in both Broker instances to specify the transport address of the load Broker. The unload Broker must know the address of the load broker, and the load Broker must in turn know the address of the unload Broker.

Example:

Broker ETB001 performs the unload on host HOST1, and Broker ETB002 performs the load on host HOST2. The transmission is based on TCP/IP. Therefore, Broker ETB001 starts the TCP/IP communicator to establish port 1971, and Broker ETB002 starts the TCP/IP communicator to establish port 1972.

For ETB001, attribute `PARTNER-CLUSTER-ADDRESS=HOST2:1972:TCP` is set, and for ETB002, attribute `PARTNER-CLUSTER-ADDRESS=HOST1:1971:TCP` is set to establish the Broker-to-Broker communication between the two Broker instances.

In addition to attributes RUN-MODE and PARTNER-CLUSTER-ADDRESS, a fully functioning Broker configuration is required when starting the two Broker instances. To access an existing PSTORE on the unloader side, you must set the attribute PSTORE=HOT. To load the data into the new PSTORE on the loader side, you must set the attribute PSTORE=COLD. The load process requires an empty PSTORE at the beginning of the load process.

Note: Use caution not to assign PSTORE=COLD to your unload Broker instance, as this startup process will erase all data currently in the PSTORE.

For the migration process, the unload Broker and the load Broker must be assigned different persistent stores.

A report can be generated to detail all of the contents of the existing persistent store. At the end of the migration process, a second report can be run on the resulting new persistent store. These two reports can be compared to ensure that all contents were migrated properly. To run these reports, set the attribute PSTORE-REPORT=YES. See PSTORE for detailed description, especially for the file assignment.

Migration Procedure

The migration procedure is made up of three steps.

Step 1

The unload Broker and the load Broker instances can be started independently of each other. Each instance will wait for the other to become available before starting the unload/load procedure.

The unload Broker instance sends a handshake request to the load Broker instance in order to perform an initial compatibility check. This validation is performed by Broker according to platform architecture type and Broker version number. The handshake ensures a correctly configured partner cluster address and ensures that the user did not assign the same PSTORE to both Broker instances. If a problem is detected, an error message will be issued and both Broker instances will stop.

Step 2

The unload Broker instance reads all PSTORE data in a special non-destructive raw mode and transmits the data to the load Broker instance. The load Broker instance writes the unchanged raw data to the new PSTORE. A report is created if PSTORE-REPORT=YES is specified, and a valid output file for the report is specified.

Step 3

The unload Broker instance requests a summary report from the load Broker instance to compare the amount of migrated data. The result of this check is reported by the unload Broker instance and the load Broker instance before they shut down.

When a Broker instances is started in `RUN-MODE=PSTORE-LOAD` or `RUN-MODE=PSTORE-UNLOAD`, the Broker instances only allow Administration requests. All other user requests are prohibited.

Notes:

1. The contents of the persistent store are copied to the new persistent store as an exact replica. No filtering of unnecessary information will be performed, for example, UOWs in received state. The master records will not be updated.
2. Before restarting your Broker with the new persistent store, be sure to change your PSTORE attribute to `PSTORE=HOT`. *Do not* start your broker with the new persistence store using `PSTORE=COLD`; this startup process will erase all of the data in your persistent store.
3. After completing the migration process and restarting your broker in a normal run-mode, it is important not to bring both the new PSTORE and the old PSTORE back online using separate Broker instances; otherwise, applications would receive the same data twice. Once the migration process is completed satisfactorily, and is validated, the old PSTORE contents should be discarded.

7 Tracing EntireX Components under z/VSE

Tracing EntireX Broker

To switch on tracing

- Set the attribute `TRACE-LEVEL` in the broker attribute file
  - for minimal trace output to "1"
  - for detailed trace output to "2"
  - for full trace output to "3"

  Example:

```
TRACE-LEVEL=2
```

To switch off tracing

- Set the attribute `TRACE-LEVEL` in the broker attribute file to 0:

```
TRACE-LEVEL=0
```

  Or:

  Omit the `TRACE-LEVEL` attribute.

Tracing Broker Stubs

The broker stubs provide an option for writing trace files. Trace output is only available for transport method TCP.

To switch on tracing for the broker stub

- Before starting the client application, set the environment variable `STUBLOG` in your application job control or CICS startup:

Trace Value	Trace Level	Description
0	NONE	No tracing.
1	STANDARD	Traces initialization, errors, and all ACI request/reply strings.
2	ADVANCED	Used primarily by system engineers, traces everything from level 1 and provides additional information - for example the Broker ACI control block - as well as transport information.
3	SUPPORT	This is full tracing through the stub, including detailed traces of control blocks, message information, etc.

Example:

```
//SETPARM STUBLOG=2
```

If the trace level is greater than 1, unencrypted contents of the send/receive buffers may be exposed in the trace.

Trace output is written to SYSOUT.

Remember to switch off tracing to prevent trace files from filling up your disk.

To switch off tracing for the broker stub

- Set the environment variable STUBLOG to NONE or remove the SETPARM statement.

Activating Tracing for the RPC Server

To switch on tracing for the RPC server

- Set the parameter TRACELEVEL in the configuration file RPCPARM in sublibrary EXP960. Example:

  ```
  TRACELEVEL=SUPPORT
  ```

  To evaluate the return codes, see *Error Messages and Codes*.

8 Broker Shutdown Statistics

Shutdown Statistics Output

After a successful Broker execution, shutdown statistics and related information are produced. This output is written in the following sequence:

1. The diagnostic message ETBD0444 is written into the Broker trace log.
2. The output - i.e. statistics, internals and user-specified parameters - is written into the end of the Broker trace log file at shutdown.

Table of Shutdown Statistics

See *Legend* below for explanation of output type.

Output Type	Display Field	Description
U	Broker ID	Identifies the Broker kernel to which the attribute file applies. See BROKER-ID.
I	Version	The version of the Broker kernel currently running.
I	Generated platform family	The platform family for which this Broker kernel was built.
I	Runtime platform	The platform on which this Broker kernel is currently running.
I	Start time	The date and time when this Broker kernel started.
S	Restart count	The restart count indicates how many times the Broker kernel has been started with the persistent store. Therefore, after a cold start (PSTORE=COLD), the restart count will be 1. Then, after subsequent hot starts (PSTORE=HOT), the restart count will be 2 or greater.
U	Trace level	The value for the trace setting for this Broker kernel. See TRACE-LEVEL.
U	Worker tasks	The number of worker tasks for this Broker kernel. See NUM-WORKER.
U	MAX-MEMORY	The value of MAX-MEMORY or 0 if not defined. See MAX-MEMORY.
S	Memory allocated	Size of the allocated memory, in bytes.
S	Memory allocated HWM	Highest size of allocated memory in bytes since Broker started.
U	NUM-SERVICE	Value of NUM-SERVICE or 0 if not defined. See NUM-SERVICE.
S	Services active	The number of services currently active for this Broker kernel.
U	NUM-CLIENT	Value of NUM-CLIENT or 0 if not defined. See NUM-CLIENT.
S	Clients active	The number of clients currently active for this Broker kernel.
S	Clients active HWM	The high watermark for the number of clients active for this Broker kernel.

Output Type	Display Field	Description
U	NUM-SERVER	Value of NUM-SERVER or 0 if not defined. See NUM-SERVER.
S	Servers active	The number of servers currently active for this Broker kernel.
S	Servers active HWM	The high watermark for the number of servers active for this Broker kernel.
U	NUM-CONVERSATION	Value of NUM-CONVERSATION or 0 if not defined. See NUM-CONVERSATION.
S	Conversations active	The number of conversations currently active for this Broker kernel.
S	Conversations active HWM	The high watermark for the number of conversations active for this Broker kernel.
U	NUM-LONG-BUFFER	Value of NUM-LONG-BUFFER or 0 if not defined. See NUM-LONG-BUFFER.
S	Long buffers active	The number of long message buffers currently in use for this Broker kernel.
S	Long buffers active HWM	The high watermark for the number of long message buffers used for this Broker kernel.
U	NUM-SHORT-BUFFER	Value of NUM-SHORT-BUFFER or 0 if not defined. See NUM-SHORT-BUFFER.
S	Short buffers active	The number of short message buffers currently in use for this Broker kernel.
S	Short buffers active HWM	The high watermark for the number of short message buffers used for this Broker kernel.
U	NUM-TOPIC	Value of NUM-TOPIC or 0 if not defined. See NUM-TOPIC.
S	Topics active	The number of topics currently active for this Broker kernel.
U	NUM-PUBLISHER	Value of NUM-PUBLISHER or 0 if not defined.
S	Publishers active	The number of publishers currently active for this Broker kernel.
S	Publishers active HWM	The high watermark for the number of publishers active for this Broker kernel.
U	NUM-SUBSCRIBER	Value of NUM-SUBSCRIBER or 0 if not defined. See NUM-SUBSCRIBER.
S	Subscribers active	The number of subscribers currently active for this Broker kernel.
S	Subscribers active HWM	The high watermark for the number of subscribers active for this Broker kernel.
U	NUM-PUBLICATION	Value of NUM-PUBLICATION or 0 if not defined. See NUM-PUBLICATION.
S	Publications active	The number of publications currently active for this Broker kernel.
S	Publications active HWM	The high watermark for the number of publications active for this Broker kernel.

Output Type	Display Field	Description
U	Persistent store type	The type of persistent store used by this Broker kernel. See PSTORE-TYPE.
U	UOW persistence	Indicates whether units of work are persistent or not in this Broker kernel. See STORE.
U	Persistent store startup	Indicates the status of the persistent store at Broker startup. See PSTORE.
U	Persistent status lifetime	The multiplier to compute the lifetime of the persistent status. See UWSTATP.
U	Deferred UOWs allowed	Indicates whether or not deferred units of work are allowed. See DEFERRED.
U	Maximum allowed UOWs	The maximum number of units of work that can be active concurrently for this Broker kernel. See MAX-UOWS.
U	Maximum messages per UOW	The maximum number of messages allowed in a unit of work. See MAX-MESSAGES-IN-UOW.
U	UOW lifetime in seconds	Indicates the default lifetime for a unit of work. See UWTIME.
U	Maximum message length	Indicates the maximum message size that can be sent. See MAX-UOW-MESSAGE-LENGTH.
U	New UOW messages allowed	Indicates whether or not new units of work are allowed in this Broker kernel. See NEW-UOW-MESSAGES.
S	UOWs active	The number of units of work currently active in this Broker kernel.
S	Current UOW	The number of the last unit of work in this Broker kernel.
U	Accounting	Indicates the status of accounting records in this Broker kernel. See ACCOUNTING.
U	TCP port *	If applicable, the TCP port number on which this Broker kernel will listen for connection requests. See TCPPORT.
I	Number of function calls	Marks the beginning of the section of summary statistics for all the function calls.
S	DEREGISTER	The number of Broker DEREGISTER function calls since startup.
S	EOC	The number of Broker EOC function calls since startup.
S	KERNELVERS	The number of Broker KERNELVERS function calls since startup.
S	LOGOFF	The number of Broker LOGOFF function calls since startup.
S	LOGON	The number of Broker LOGON function calls since startup.
S	RECEIVE	The number of Broker RECEIVE function calls since startup.
S	REGISTER	The number of Broker REGISTER function calls since startup.
S	SEND	The number of Broker SEND function calls since startup.
S	SYNCPOINT	The number of Broker SYNCPOINT function calls since startup.
S	UNDO	The number of Broker UNDO function calls since startup.

Output Type	Display Field	Description
S	CONTROL_PUBLICATION	The number of Broker CONTROL_PUBLICATION function calls since startup.
S	RECEIVE_PUBLICATION	The number of Broker RECEIVE_PUBLICATION function calls since startup.
S	SEND_PUBLICATION	The number of Broker SEND_PUBLICATION function calls since startup.
S	SUBSCRIBE	The number of Broker SUBSCRIBE function calls since startup.
S	UNSUBSCRIBE	The number of Broker UNSUBSCRIBE function calls since startup.
S	REPLY_ERROR	The number of Broker REPLY_ERROR function calls since startup.
I	Worker task statistics	Marks the beginning of the section of summary statistics for all the worker tasks.
I	Worker number	The identifier of the worker task.
I	Status	The status of the worker task at shutdown.
S	# of calls	The number of Broker calls handled by the worker task since startup.
S	Idle time in seconds	The number of seconds the worker task has been idle since startup.

* Does not apply to z/OS.

Legend

Output Type	Description	Value	Origin of Value
I	Internal Information	Static	Determined by Software AG EntireX.
S	Shutdown Statistic	Variable	Determined by Broker activity during execution.
U	User-Specified Parameter	Variable	Specified by Broker administrator before or, if allowable, during execution.

9 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* in the platform-independent administration documentation.

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.

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 topic name and user ID.

Use the System Management Hub to define a filter. . During processing, the Broker evaluates the class, server, service, topic, 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.

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.

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`.

10 Accounting in EntireX Broker

This chapter describes the accounting records for Broker that can be used for several purposes, including:

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

EntireX Accounting Data Fields

In the EntireX Accounting record, there are various types of data available for consumption by applications that process the accounting data:

Field Name	Accounting Version	Type of Field	Description
Record Write Time	1	A14 Timestamp in "YYYYMMDDHHMMSS" format	The time this record was written to the accounting file in YYYYMMDDHHMMSS format.
EntireX Broker ID	1	A32	Broker ID from attribute file.
EntireX Version	1	A8	Version information, *v.r.s.p* where *v* =version *r* =release *s* =service pack *p* =patch level for example 9.6.0.00.
Platform of Operation	1	A32	Platform where EntireX is running.
EntireX Start Time	1	A14 Timestamp in "YYYYMMDDHHMMSS" format	Time EntireX was initialized in YYYYMMDDHHMMSS format.
Accounting Record Type	1	A1	It is always C for conversation. Future Types will have a different value in this field.
Client User ID	1	A32	USER-ID ACI field from the client in the conversation.
Client Token	1	A32	`TOKEN` field from the ACI from the client.

Field Name	Accounting Version	Type of Field	Description
Client Physical ID	1	A56	The physical user ID of the client, set by EntireX.
Client Communication Type	1	I1	Communication used by client: 1 = Net-Work 2 = TCP/IP 3 = APPC 4 = WebSphere MQ 5 = SSL
Client Requests Made	1	I4	Number of Requests made by client.
Client Sent Bytes	1	I4	Number of bytes sent by client.
Client Received Bytes	1	I4	Number of bytes received by client.
Client Sent Messages	1	I4	Number of messages sent by client.
Client Received Messages	1	I4	Number of messages received by client.
Client Sent UOWs	1	I4	Number of UOWs sent by client.
Client UOWs Received	1	I4	Number of UOWs received by client.
Client Completion Code	1	I4	Completion code client received when conversation ended.
Server User ID	1	A32	`USER-ID` ACI field from the server in the conversation.
Server Token	1	A32	`TOKEN` field from the ACI from the server.
Server Physical ID	1	A56	The physical user ID of the server, set by EntireX.
Server Communication Type	1	I1	Communication used by Server: 1 = Entire Net-Work 2 = TCP/IP 3 = APPC 4 = WebSphere MQ 5 = SSL
Server Requests Made	1	I4	Number of requests made by server.
Server Sent Bytes	1	I4	Number of bytes sent by server.
Server Received Bytes	1	I4	Number of bytes received by server.
Server Sent Messages	1	I4	Number of messages sent by server.
Server Received Messages	1	I4	Number of messages received by server.
Server Sent UOWs	1	I4	Number of UOWs sent by server.
Server Received UOWs	1	I4	Number of UOWs received by server.

Field Name	Accounting Version	Type of Field	Description
Server Completion Code	1	I4	Completion code server received when conversation ended.
Conversation ID	1	A16	`CONV-ID` from ACI.
Server Class	1	A32	`SERVER-CLASS` from ACI.
Server Name	1	A32	`SERVER-NAME` from ACI.
Service Name	1	A32	`SERVICE` from ACI.
CID=NONE Indicator	1	A1	Will be N if `CONV-ID=NONE` is indicated in application.
Restarted Indicator	1	A1	Will be R if a conversation was restarted after a Broker shutdown.
Conversation Start Time	1	A14 Timestamp in "YYYYMMDDHHMMSS" format	Time conversation began in YYYYMMDDHHMMSS format.
Conversation End Time	1	A14 Timestamp in "YYYYMMDDHHMMSS" format	Time conversation was cleaned up in YYYYMMDDHHMMSS format.
Conversation CPU Time	1	I4	Number of microseconds of CPU time used by the conversation
Client Security Identity	2	A32	Actual identity of client derived from authenticated user ID.
Client Application Node	2	A32	Node name of machine where client application executes.
Client Application Type	2	A8	Stub type used by client application.
Client Application Name	2	A64	Name of the executable that called the broker. Corresponds to the Broker Information Service field *APPLICATION-NAME* in the ACI Programming documentation.
Client Credentials Type	2	I1	Mechanism by which authentication is performed for client.
Server Security Identity	2	A32	Actual identity of server derived from authenticated user ID.
Server Application Node	2	A32	Node name of machine where server application executes.
Server Application Type	2	A8	Stub type used by server application.
Server Application Name	2	A64	Name of the executable that called the broker. Corresponds to the Broker Information Service field *APPLICATION-NAME* in the ACI Programming documentation.

Field Name	Accounting Version	Type of Field	Description
Server Credentials Type	2	I1	Mechanism by which authentication is performed for server.
Client RPC Library	3	A128	RPC library referenced by client when sending the only/first request message of the conversation.
Client RPC Program	3	A128	RPC Program referenced by client when sending the only/first request message of the conversation.
Server RPC Library	3	A128	RPC library referenced by server when sending the only/first response message of the conversation.
Server RPC Program	3	A128	RPC Program referenced by server when sending the only/first response message of the conversation.
Client IPv4 Address	4	A16	IPv4 address of the client.
Server IPv4 Address	4	A16	IPv4 address of the server.
Client Application Version	4	A16	Application version of the client.
Server Application Version	4	A16	Application version of the server.
Client IPv6 Address	5	A46	IPv6 address of the client.
Server IPv6 Address	5	A46	IPv6 address of the server.

Note: Accounting fields of any version greater than 1 are created only if the attribute `ACCOUNTING-VERSION` value is greater than or equal to the corresponding version. For example: accounting fields of version 2 are visible only if `ACCOUNTING-VERSION=2` or higher is specified.

Example Uses of Accounting Data

- Chargeback
- Trend Analysis

- Tuning for Application Performance

Chargeback

Customers can use the EntireX accounting data to perform chargeback calculations for resource utilization in a data center. Suppose EntireX Broker is being used to dispatch messages for three business departments: Accounts Receivable, Accounts Payable, and Inventory. At the end of each month, the customer needs to determine how much of the operation and maintenance cost of EntireX Broker should be assigned to these departments. For a typical month, assume the following is true:

Department	Amount of Data	Percentage	Messages Sent	Percentage	Average Percentage
Accts Payable	50 MB	25	4000	20	22.5
Accts Receivable	40 MB	20	6000	30	25
Inventory	110 MB	55	10000	50	52.5

The use of Broker resources here is based upon both the amount of traffic sent to the Broker (bytes) as well as how often the Broker is called (messages). The average of the two percentages is used to internally bill the departments, so 52.5% of the cost of running EntireX Broker would be paid by the Inventory Department, 25% by the Accounts Receivable Department, and 22.5% by the Accounts Payable Department.

Trend Analysis

The Accounting Data can also be used for trend analysis. Suppose a customer has several point-of-sale systems in several stores throughout the United States that are tied into the corporate inventory database with EntireX. The stubs would be running at the stores, and the sales data would be transmitted to the Broker, which would hand it off to the appropriate departments in inventory. If these departments wish to ascertain when the stores are busiest, they can use the accounting data to monitor store transactions. Assume all of the stores are open every day from 9 AM to 10 PM.

Local Time	Average: Weekday Transactions per Store	Maximum Weekday Transactions in any Store	Average Weekend Transactions per Store	Maximum Weekend Transactions in any Store
9 AM	7.3	27	28.2	83
10 AM	11.2	31	29.3	102
11 AM	14.6	48	37.9	113
12 noon	56.2	106	34.8	98
1 PM	25.6	65	34.2	95
2 PM	17.2	52	38.5	102
3 PM	12.1	23	42.7	99

Local Time	Average: Weekday Transactions per Store	Maximum Weekday Transactions in any Store	Average Weekend Transactions per Store	Maximum Weekend Transactions in any Store
4 PM	18.3	34	43.2	88
5 PM	26.2	47	45.2	93
6 PM	38.2	87	40.6	105
7 PM	29.6	83	39.2	110
8 PM	18.6	78	28.6	85
9 PM	11.2	55	17.5	62

The owner of the stores can examine the data and make decisions based upon the data here. For example, on weekdays, he or she can see that there is little business until lunchtime, when the number of transactions increase. It then decreases during lunch hour; then there is another increase from 5 PM to 8 PM, after people leave work. Based on this data, the owner might investigate changing the store hours on weekdays to 10 AM to 9 PM. On the weekend the trends are different, and the store hours could be adjusted as well, although there is a more regular customer flow each hour on the weekends.

Tuning for Application Performance

Assume that a customer has two applications that perform basic request/response messaging for similar sized messages. The applications consist of many Windows PC clients and Natural RPC Servers on UNIX. An analysis of the accounting data shows the following:

Application Type	Class	Server	Service	Average Server Messages Received per Conversation	Average Client Messages Received per Conversation
Application 1:	CLASS1	SERVER1	SERVICE1	10.30	10.29
Application 2:	CLASS2	SERVER2	SERVICE2	10.30	8.98

A further analysis of the accounting data reveals that there are a lot of non-zero response codes in the records pertaining to Application 2, and that a lot of these non-zero responses indicate timeouts. With that information, the customer can address the problem by modifying the server code, or by adjusting the timeout parameters for SERVER2 so that it can have more time to get a response from the Service.

11 Broker Resource Allocation

The EntireX Broker is a multithreaded application and communicates among multiple tasks in memory pools.

General Considerations

Resource considerations apply to both the global and service-specific levels:

- Dynamic assignment of global resources to services that need them prevents the return of a "Resource Shortage" code to an application when resources are available globally. It also enables the EntireX Broker to run with fewer total resources, although it does not guarantee the availability of a specific set of resources for a particular service.
- Flow control ensures that individual services do not influence the behavior of other services by accident, error, or simply overload. This means that you can restrict the resource consumption of particular services in order to shield the other services.

In order to satisfy both global and service-specific requirements, the EntireX Broker allows you to allocate resources for each individual service or define global resources which are then allocated dynamically to any service that needs them.

The resources in question are the number of conversations, number of servers, plus units of work and the message storage, separated in a long buffer of 4096 bytes and short buffer of 256 bytes. These resources are typically the bottleneck in a system, especially when you consider that non-conversational communication is treated as the special case of "conversations with a single message only" within the EntireX Broker.

Global resources are defined by the parameters in the Broker section of the attribute file. The number of conversations allocated to each service is defined in the service-specific section of the attribute file. Because the conversations are shared by all servers that provide the service, a larger number of conversations should be allocated to services that are provided by more than one server. The number of conversations required is also affected by the number of clients accessing the service in parallel.

Specifying Global Resources

You can specify a set of global resources with no restrictions on which service allocates the resources:

- Specify the global attributes with the desired values.
- Do not specify any additional restrictions. That is, do not provide values for the following Broker-specific attributes:

```
LONG-BUFFER-DEFAULT
SHORT-BUFFER-DEFAULT
```

```
CONV-DEFAULT
SERVER-DEFAULT
```

- Also, do not provide values for the following server-specific attributes:

```
LONG-BUFFER-LIMIT
SERVER-LIMIT
SHORT-BUFFER-LIMIT
CONV-LIMIT
```

Example

The following example defines global resources. If no additional definitions are specified, resources are allocated and assigned to any server that needs them.

```
NUM-CONVERSATION=1000
NUM-LONG-BUFFER=200
NUM-SHORT-BUFFER=2000
NUM-SERVER=100
```

Restricting the Resources of Particular Services

You can restrict resource allocation for particular services in advance:

- Use `CONV-LIMIT` to limit the resource consumption for a specific service.
- Use `CONV-DEFAULT` to provide a default limit for services for which `CONV-LIMIT` is not defined.

Example

In the following example, attributes are used to restrict resource allocation:

```
DEFAULTS=BROKER
NUM-CONVERSATION=1000
CONV-DEFAULT=200

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, CONV-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B, CONV-LIMIT=UNLIM
CLASS=C, SERVER=C, SERVICE=C
```

- Memory for a total of 1000 conversions is allocated (`NUM-CONVERSATION=1000`).
- Service A (`CLASS A`,`SERVER A`,`SERVICE A`) is limited to 100 conversation control blocks used simultaneously (`CONV-LIMIT=100`). The application that wants to start more conversations than specified by the limit policy will receive a “Resource shortage” return code. This return code should result in a retry of the desired operation a little later, when the resource situation may have changed.

- Service B (CLASS B,SERVER B,SERVICE B) is allowed to try to allocate as many resources as necessary, provided the resources are available and not occupied by other services. The number of conversations that may be used by this service is unlimited (CONV-LIMIT=UNLIM).
- Service C (CLASS C,SERVER C,SERVICE C) has no explicit value for the CONV-LIMIT attribute. The number of conversation control blocks that it is allowed to use is therefore limited to the default value which is defined by the CONV-DEFAULT Broker attribute.

The same scheme applies to the allocation of message buffers and servers:

- In the following example, long message buffers are allocated using the keywords NUM-LONG-BUFFER, LONG-BUFFER-DEFAULT and LONG-BUFFER-LIMIT:

```
DEFAULTS=BROKER
NUM-LONG-BUFFER=2000
LONG-BUFFER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, LONG-BUFFER-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B, LONG-BUFFER-LIMIT=UNLIM
CLASS=C, SERVER=C, SERVICE=C
```

- In the following example, short message buffers are allocated using the keywords NUM-SHORT-BUFFER, SHORT-BUFFER-DEFAULT and SHORT-BUFFER-LIMIT:

```
DEFAULTS=BROKER
NUM-SHORT-BUFFER=2000
SHORT-BUFFER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, SHORT-BUFFER-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B, SHORT-BUFFER-LIMIT=UNLIM
CLASS=C, SERVER=C, SERVICE=C
```

- In the following example, servers are allocated using the keywords NUM-SERVER, SERVER-DEFAULT and SERVER-LIMIT:

```
DEFAULTS=BROKER
NUM-SERVER=2000
SERVER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, SERVER-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B, SERVER-LIMIT=UNLIM
CLASS=C, SERVER=C, SERVICE=C
```

Specifying Attributes for Privileged Services

If privileged services (services with access to unlimited resources) exist, specify `UNLIMITED` for the attributes `CONV-LIMIT`, `SERVER-LIMIT`, `LONG-BUFFER-LIMIT` and `SHORT-BUFFER-LIMIT` in the service-specific section of the attribute file.

For example:

```
DEFAULTS=SERVICE
CONV-LIMIT=UNLIM
LONG-BUFFER-LIMIT=UNLIM
SHORT-BUFFER-LIMIT=UNLIM
SERVER-LIMIT=UNLIM
```

To ensure a resource reservoir for peak load of privileged services, define more resources than would normally be expected by specifying larger numbers for the Broker attributes that control global resources:

```
NUM-SERVER
NUM-CONVERSATION
CONV-DEFAULT
LONG-BUFFER-DEFAULT
SHORT-BUFFER-DEFAULT
SERVER-DEFAULT
```

Maximum Units of Work

The maximum number of units of work (UOWs) that can be active concurrently is specified in the Broker attribute file. The `MAX-UOWS` attribute can be specified for the Broker globally as well as for individual services. It cannot be calculated automatically. If a service is intended to process UOWs, a `MAX-UOWS` value must be specified.

If message processing only is to be done, specify `MAX-UOWS=0` (zero). The Broker (or the service) will not accept units of work, i.e., it will process only messages that are not part of a UOW. Zero is used as the default value for `MAX-UOWS` in order to prevent the sending of UOWs to services that are not intended to process them.

Calculating Resources Automatically

To ensure that each service runs without impacting other services, allow the EntireX Broker to calculate resource requirements automatically:

- Ensure that the attributes that define the default total for the Broker and the limit for each service are not set to `UNLIM`.
- Specify `AUTO` for the Broker attribute that defines the total number of the resource.
- Specify a suitable value for the Broker attribute that defines the default number of the resource.

The total number required will be calculated from the number defined for each service. The resources that can be calculated this way are Number of Conversations, Number of Servers, Long Message Buffers and Short Message Buffers.

Avoid altering the service-specific definitions at runtime. Doing so could corrupt the conversation consistency. Applications might receive a message such as "`NUM-CONVERSATIONS` reached" although the addressed service does not serve as many conversations as defined. The same applies to the attributes that define the long and short buffer resources.

Automatic resource calculation has the additional advantage of limiting the amount of memory used to run the EntireX Broker. Over time, you should be able to determine which services need more resources by noting the occurrence of the return code "resource shortage, please retry". You can then increase the resources for these services. To avoid disruption to the user, you could instead allocate a relatively large set of resources initially and then decrease the values using information gained from the Administration Monitor application.

Number of Conversations

To calculate the total number of conversations automatically, ensure that the `CONV-DEFAULT` Broker attribute and the `CONV-LIMIT` service-specific attribute are not set to `UNLIM` anywhere in the attribute file. Specify `NUM-CONVERSATION=AUTO` and an appropriate value for the `CONV-DEFAULT` Broker attribute. The total number of conversations will be calculated using the value specified for each service.

For example:

```
DEFAULTS=BROKER
NUM-CONVERSATION=AUTO
CONV-DEFAULT=200

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A
CLASS=B, SERVER=B, SERVICE=B, CONV-LIMIT=100
CLASS=C, SERVER=C, SERVICE=C
```

- Service A and Service C both need 200 conversations (the default value). Service B needs 100 conversations (CONV-LIMIT=100).
- Because NUM-CONVERSATIONS is defined as AUTO, the broker calculates a total of 500 conversations (200 + 200 + 100).
- NUM-CONVERSATIONS=AUTO allows the number of conversations to be flexible without requiring additional specifications. It also ensures that the broker is started with enough resources to meet all the demands of the individual services.
- AUTO and UNLIM are mutually exclusive. If CONV-DEFAULT or a single CONV-LIMIT is defined as UNLIM, the EntireX Broker cannot determine the number of conversations to use in the calculation, and the EntireX Broker cannot be started.

Number of Servers

To calculate the number of servers automatically, ensure that the SERVER-DEFAULT Broker attribute and the SERVER-LIMIT service-specific attribute are not set to UNLIM anywhere in the attribute file. Specify NUM-SERVER=AUTO and an appropriate value for the SERVER-DEFAULT Broker attribute. The total number of server buffers will be calculated using the value specified for each service.

For example:

```
DEFAULTS=BROKER
NUM-SERVER=AUTO
SERVER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, SERVER-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B
CLASS=C, SERVER=C, SERVICE=C
```

Long Message Buffers

To calculate the number of long message buffers automatically, ensure that the LONG-BUFFER-DEFAULT Broker attribute and the LONG-BUFFER-LIMIT service-specific attribute are not set to UNLIM anywhere in the attribute file. Specify NUM-LONG-BUFFER=AUTO and an appropriate value for the LONG-BUFFER-DEFAULT Broker attribute. The total number of long message buffers will be calculated using the value specified for each service.

For example:

```
DEFAULTS=BROKER
NUM-LONG-BUFFER=AUTO
LONG-BUFFER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A, LONG-BUFFER-LIMIT=100
CLASS=B, SERVER=B, SERVICE=B
CLASS=C, SERVER=C, SERVICE=C
```

Short Message Buffers

To calculate the number of short message buffers automatically, ensure that the `SHORT-BUFFER-DEFAULT` Broker attribute and the `SHORT-BUFFER-LIMIT` service-specific attribute are not set to `UNLIM` anywhere in the attribute file. Specify `NUM-SHORT-BUFFER=AUTO` and an appropriate value for the `SHORT-BUFFER-DEFAULT` Broker attribute. The total number of short message buffers will be calculated using the value specified for each service.

For example:

```
DEFAULTS=BROKER
NUM-SHORT-BUFFER=AUTO
SHORT-BUFFER-DEFAULT=250

DEFAULTS=SERVICE
CLASS=A, SERVER=A, SERVICE=A
CLASS=B, SERVER=B, SERVICE=B, SHORT-BUFFER-LIMIT=100
CLASS=C, SERVER=C, SERVICE=C
```

Dynamic Memory Management

Dynamic memory management is a feature to handle changing Broker workload without any restart of the Broker task. It increases the availability of the Broker by using various memory pools for various Broker resources and by being able to use a variable number of pools for the resources.

If more memory is needed than currently available, another memory pool is allocated for the specific type of resource. If a particular memory pool is no longer used, it will be deallocated.

The following Broker attributes can be omitted if `DYNAMIC-MEMORY-MANAGEMENT=YES` has been defined:

- NUM-CLIENT
- NUM-CMDLOG-FILTER
- NUM-COMBUF
- NUM-CONV[ERSATION]
- NUM-LONG[-BUFFER]
- NUM-PUBLICATION
- NUM-PUBLISHER
- NUM-SERVER
- NUM-SERVICE
- NUM-SERVICE-EXTENSION
- NUM-SHORT[-BUFFER]
- NUM-SUBSCRIBER
- NUM-SUBSCRIBER-TOTAL
- NUM-TOPIC
- NUM-TOPIC-EXTENSION
- NUM-TOPIC-TOTAL
- NUM-UOW|MAX-UOWS|MUOW
- NUM-WQE

If you want statistics on allocation and deallocation operations in Broker, you can configure Broker to create a storage report with the attribute STORAGE-REPORT. See ***Storage Report*** below.

Note: To ensure a stabile environment, some pools of Broker are not deallocated automatically. The first pools of type COMMUNICATION, CONVERSATION, CONNECTION, HEAP, PARTICIPANT, PARTICIPANT EXTENSION, SERVICE ATTRIBUTES, SERVICE, SERVICE EXTENSION, TIMEOUT QUEUE, TRANSLATION, WORK QUEUE are excluded from the automatic deallocation even when they have not been used for quite some time. Large pools cannot be reallocated under some circumstances if the level of fragmentation in the address space has been increased in the meantime.

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

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
SUBSCRIPTION POOL         0x27A15010    344148 bytes   29401856 bytes  2009-06-26 ↩
12:28:58.839  Allocated
TOPIC ATTRIBUTES POOL     0x27A6A010    129620 bytes   29531476 bytes  2009-06-26 ↩
12:28:58.841  Allocated
TOPIC POOL                0x26FB6068      2952 bytes   29534428 bytes  2009-06-26 ↩
12:28:58.842  Allocated
TOPIC EXTENSION POOL      0x27A8A010     30852 bytes   29565280 bytes  2009-06-26 ↩
12:28:58.842  Allocated
PSTORE SUBSCRIBER POOL    0x27A92010     33892 bytes   29599172 bytes  2009-06-26 ↩
12:28:58.843  Allocated
PSTORE TOPIC POOL         0x27A9B010     19540 bytes   29618712 bytes  2009-06-26 ↩
12:28:58.843  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
SUBSCRIPTION POOL         0x27A15010    344148 bytes    2941156 bytes  2009-06-26 ↩
12:30:58.530  Deallocated
TOPIC ATTRIBUTES POOL     0x27A6A010    129620 bytes    2811536 bytes  2009-06-26 ↩
12:30:58.531  Deallocated
TOPIC POOL                0x26FB6068      2952 bytes    2808584 bytes  2009-06-26 ↩
12:30:58.531  Deallocated
TOPIC EXTENSION POOL      0x27A8A010     30852 bytes    2777732 bytes  2009-06-26 ↩
12:30:58.531  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
PSTORE SUBSCRIBER POOL    0x27A92010     33892 bytes    2046896 bytes  2009-06-26 ↩
12:30:58.534  Deallocated
PSTORE TOPIC POOL         0x27A9B010     19540 bytes    2027356 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.

Maximum TCP/IP Connections per Communicator

This table shows the maximum number of TCP/IP connections per communicator:

Platform	Maximum Number of TCP/IP Connections per Communicator
AIX	2,048
BS2000/OSD	2,048
HP-UX	2,048
Linux	4,096
Solaris	65,356
Windows	4,096
z/OS	16,384
z/VSE	2,048

With the Broker-specific attribute `POLL`, these restrictions can be lifted under z/OS, UNIX and z/VSE. See `POLL`.

See also `MAX-CONNECTIONS` under *`TCP-OBJECT (Struct INFO_TCP)`* under *Information Reply Structures* in the Broker CIS documentation.