Setting up Broker Instances

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

- Setting up TCP/IP Transport
- Running Broker with SSL or TLS Transport
- Setting up Entire Net-Work/Adabas SVC Transport
- Starting and Stopping the Broker
- Dual Command Log Files
- Tracing EntireX Broker
- Protecting a Broker against Denial-of-Service Attacks
- Setting the Time Zone for Broker

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|hostname` and `STACK-NAME=stackname` under *TCP/IP-specific Attributes* under *Broker Attributes*.

However, if no port number is specified in the broker attribute file, the EntireX Broker kernel uses `getservbyname` to determine the TCP/IP port on which it will listen for incoming connections. The specified name is the value of `BROKER-ID` in the attribute file. An entry for this value must be made in the local machine's */etc/services* file. Example:

```
ETBnnn yyyyy/tcp # local host
```

where *etbnnn* is the `BROKER-ID` and *yyyyy* is the intended port number. This is the same place from which local Broker stubs will obtain the port information. If `getservbyname` fails, then a default port number of 1971 will be used. This is the same default port number that the stubs use.

Running Broker with SSL or TLS Transport

Before starting the Broker, it must be configured to correctly use SSL or TLS as a transport mechanism:

- Step 1: Modify Broker-specific Attributes
- Step 2: Modify SSL-specific Attributes

Step 1: Modify Broker-specific Attributes

Append "-SSL" to the TRANSPORT attribute. For example:

```
DEFAULTS = BROKER
TRANSPORT = TCP-SSL
```

See also TRANSPORT.

Step 2: Modify SSL-specific Attributes

Set the SSL or TLS attributes, for example:

```
DEFAULTS = SSL
 TRUST-STORE   = MYRING
 KEY-LABEL     = MYCERTIFICATE
 VERIFY-CLIENT = N
 PORT=1958
```

where *1958* is the default but can be changed to any port number.

See also *SSL-specific Attributes* under *Broker Attributes* and *SSL or TLS and Certificates with EntireX*.

Setting up Entire Net-Work/Adabas SVC Transport

To set up EntireX Net-Work communication mechanism

1. Ensure that all load libraries in the Broker kernel steplib are APF-authorized.
2. 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* under *Broker Attributes*.
3. Ensure that communication with the EntireX Broker is possible by running the installation verification programs (bcoc, bcos) using transport type NET.

Starting and Stopping the Broker

To start the Broker

1. Create a user ID for the started task or job where your Broker kernel is going to run.
2. If you are using a started task, ensure that the user ID is defined in the list of user IDs for started tasks and that the sample Broker kernel JCL is modified appropriately to create a PROC.
3. Start the Broker kernel either from the Broker kernel job (JCL) or started task (PROC).

To stop the Broker

- Issue the operator command `P <JOBNAME>`

  Or:
  Execute the `ETBCMD` utility using the example syntax below:

```
//ETBCMD  EXEC PGM=ETBCMD,
// PARM=('/-bhost:port:TCP ',
//       '-cSHUTDOWN -dBROKER -xuid -ypwd')
```

See *Operator Commands* for a full list and also *Broker Command-line Utilities*.

Dual Command Log Files

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.

Broker uses two command log files, enabling data to be written to one of the files while the other is being copied for archival purposes. Two file names must be specified for the dual command logs. At startup, Broker initializes both files and keeps the first open for printing command log data. Broker kernel switches to the other command log when the first file becomes full - or when the size of the open file reaches the value optionally specified by `CMDLOG-FILE-SIZE` (specified in KB).

Note:
It is always advisable to copy the contents of a full command log file before Broker fills the subsequent command log file. Otherwise, the information in the first file (full and closed) will be overwritten.

The file requirements are two equally sized, physical sequential files defined with a record length of 121 bytes, i.e.
`DCB=(LRECL=121,RECFM=PS,BLKSIZE=nnnn)`. We recommended you allocate files with a single (primary) extent only. For example `SPACE=(CYL,(30,0))`. The minimum file size is approximately 3 cylinders of 3390 device.

Alternatively, the dual command log files can be allowed in USS HFS file system.

For more information, see *Command Logging in EntireX*.

Tracing EntireX Broker

This section covers the following topics:

- Broker TRACE-LEVEL Attribute
- Attribute File Trace Setting
- Deferred Tracing
- Flushing Trace Data to a GDG Data Set

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.

For temporary changes to TRACE-LEVEL without restarting the Broker, use the Broker command-line utility *ETBCMD*.

Individual Settings	Specified in Attribute File Section
Master trace level	DEFAULTS=BROKER
Persistent store trace level	DEFAULTS=ADABAS | CTREE | DIV (currently not available for DIV)
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 | SSL

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.
8	All of Trace Level 4, plus Adabas Persistent Store Adabas control blocks.

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".

Flushing Trace Data to a GDG Data Set

With broker-specific attributes TRMODE=WRAP and TRBUFNUM=*n*, Broker writes trace data to internal buffers instead of stderr (DD:SYSOUT). These buffers are used in round-robin mode and do not involve any I/O operation. If you need trace data for diagnostic purposes, use the operator command *FLUSH* to write the trace data from the internal buffers to a data set. A FLUSH command is performed automatically in case of error exceptions. The output data set is not readable for any other user when the broker is running. To avoid this problem, you can use a GDG (generation data group) data set as output data set. First you must allocate the GDG and define it to the broker. These preparatory steps are outlined below. The GDG name EXX.GDG is used in the examples.

Note:
GDG is supported for deferred tracing only.

- Allocating a GDG
- Defining the GDG to Broker
- Writing to the GDG Data Set

Allocating a GDG

Define the GDG with IDCAMS. This definition is needed before working with the GDG data sets. You can use the following JCL to define a GDG. The LIMIT parameter is set to 16, but may contain other values according to your needs.

```
//IDCAMS   EXEC PGM=IDCAMS,REGION=4096K
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   *
  DEFINE GENERATIONDATAGROUP -
  (NAME(EXX.GDG) -
  NOEMPTY -
  SCRATCH -
  LIMIT(16))
/*
```

Defining the GDG to Broker

The GDG data set as target for trace data can be managed without changes to the Broker JCL. However, the DD statement for such a GDG data set has to be defined as a Broker attribute in order to propagate the file characteristics to the runtime library of the IBM Language Environment.

```
TRACE-DD = "DSNAME=EXX.GDG,
            DCB=(BLKSIZE=1210,DSORG=PS,LRECL=121,RECFM=FB),
            DISP=(NEW,CATLG,CATLG),
            SPACE=(CYL,(100,10)),
            STORCLAS=SMS"
```

See TRACE-DD under *Broker Attributes*.

Writing to the GDG Data Set

After successful broker initialization, a new data set of the GDG is allocated and opened. Based on the defined GDG name EXX.GDG in the sample JCL above, data set names EXX.GDG.G0001V00, EXX.GDG.G0002V00, EXX.GDG.G0003V00 and so on will be allocated and written.

Use operator command FLUSH to write all trace data from internal buffers to the GDG data set. The data set will be closed at the end of the FLUSH processing, and the next GDG data set is allocated and opened. During broker shutdown, the GDG data set is filled with all available trace data and closed.

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.

Setting the Time Zone for Broker

Broker obtains the time zone value by reading the environment variable TZ. If not specified, Broker retrieves the assignment of TZ from the configuration file */etc/profile*.

Check your /etc/profile for an appropriate setting of environment variable TZ. The TZ value should reflect the appropriate value for your location.

Remember that the new Daylight Saving Time rule according to the Energy Policy Act in the U.S. takes effect in 2007. If you live in an area affected by the new rule, you may append it to the TZ environment variable.

For example, `TZ=EST5EDT,M4.1.0,M10.5.0` is no longer valid.
`TZ=EST5EDT,M3.2.0,M11.1.0` must be used instead.

If you don't want to change your /etc/profile, you may configure Broker's startup JCL to define environment variable `TZ`. Modify the EXEC statement thus:

```
//BROKER   EXEC PGM=ETBNUC,REGION=0M,TIME=1440,
//         PARM='ENVAR(''TZ=EST5EDT,M3.2.0,M11.1.0'')/'
```

The value of `TZ` should reflect the appropriate value for your location. Please enclose the time zone value in quotation marks and parentheses, as outlined above. The rule for daylight saving time changes can be appended after the time zone value.