Examples for EntireX Broker Tutorial

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

This chapter covers the following topics:

- Non-conversational Examples
- Conversational Examples
- Special Features
- Getting Started
- Attach Manager Interface
- Non-blocked Server

Non-conversational Examples

- Example 1: Single Request without Reply
- Example 2: Single Request with Reply

Example 1: Single Request without Reply

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

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

Client

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

Server

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

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

Coding

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

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

Example 2: Single Request with Reply

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

Client

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

Server

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

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

Coding

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

```
Server
LOGON ------> logon to Broker
```

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

Conversational Examples

- Example 3: Long Running Service - Non-blocked Client
- Example 4: Transfer Messages from Server to Client
- Example 5: Transfer Messages from Client to Server
- Example 6: Server with Multiple Parallel Conversations

Example 3: Long Running Service - Non-blocked Client

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

Client

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

Server

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

Coding

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

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

Example 4: Transfer Messages from Server to Client

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

Client

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

Server

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

Coding

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

```
         VALUE 3 conversation ended
   until ...
until ...
LOGOFF -----> logoff from Broker
```

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

Example 5: Transfer Messages from Client to Server

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

Client

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

Server

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

Coding

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

```
Server
LOGON ------> logon to Broker
REGISTER ------> offer service
```

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

Example 6: Server with Multiple Parallel Conversations

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

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

The established communication is conversational.

Client

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

Server

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

Coding

```
Client
LOGON ------> logon to Broker
repeat
   SEND,W=nS,CID=NEW ------> initiate conversation
   repeat
      SEND,W=nS,CID=1234 ------> ongoing conversation
   until ...
   EOC,CID=1234 ------> end of conversation
until ...
LOGOFF -----> logoff from Broker
```

```
Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
   RECEIVE,W=nS,CID=ANY/OLD ------> OLD if max parallel reached
   decide on ERROR-CLASS
```

```
      VALUE 0 successful response
         SEND,W=NO,CID=1234 ------> new or ongoing conversation
      VALUE 3 conversation ended
until ...
DEREGISTER ------> deregister service
LOGOFF ------> logoff from Broker
```

Special Features

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

Example 7: Send Messages with HOLD - Delayed Delivery

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

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

Client

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

Server

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

Coding

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

```
Server
LOGON ------> logon to Broker
REGISTER ------> offer service
repeat
```

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

Example 8: Remove Service while Conversations Exist

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

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

The established communication is conversational.

Client

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

Server

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

Coding

```
Client
LOGON ------> logon to Broker
SEND,W=nS,CID=NEW ------> initiate conversation
repeat
   SEND,W=nS,CID=1234 ---> ongoing after deregistration
until ...
EOC,CID=1234 -----------> end of conversation
LOGOFF -----> logoff from Broker
```

```
Server
LOGON ------> logon to Broker
repeat
   REGISTER ------> offer service
   RECEIVE,W=nS,CID=NEW ------> wait for new conversation
   DEREGISTER,OP=QUIESCE ------> deregister service
   SEND,W=NO,CID=1234 ------> acknowledge new conversation
   repeat
      RECEIVE,W=nS,CID=1234 ------> ongoing conversation
```

```
      SEND,W=NO,CID=1234 ------> reply to client
   until 3 conversation ended
until ...
LOGOFF ------> logoff from Broker
```

Example 9: Server for Multiple Services

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

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

The established communication is non-conversational.

Client

This client demonstrates a server which is able to offer multiple services. The name of the service

the message is routed to is alternately switched between Service 1 and Service 2.

Server

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

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

Coding

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

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

Getting Started

- Example 10: ACI Test Tool: Single Broker Requests
- Example 11: Model to write Client/Server Programs API Version 1
- Example 12: Model to write Client/Server programs API Version 2

Example 10: ACI Test Tool: Single Broker Requests

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

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

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

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

The established communication is non-conversational.

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

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

Client

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

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

Server

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

Coding

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

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

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

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

The established communication is non-conversational.

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

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

Client

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

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

Server

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

Coding

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

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

Attach Manager Interface

Example 13: Demonstration of the Attach Manager Interface

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

Coding

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

Non-blocked Server

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

Example 14: Single Requests without Reply - A Polling Server

Demonstration of Attach Manager Interface:

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

Example

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

Client

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

Server

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

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

Coding

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

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

Example 15: Single Requests with Reply - A Polling Server

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

Client

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

Server

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

Coding

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

```
Server
LOGON ------> logon to Broker
REGISTER ------> offer service
```

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