Concepts of Persistent Messaging

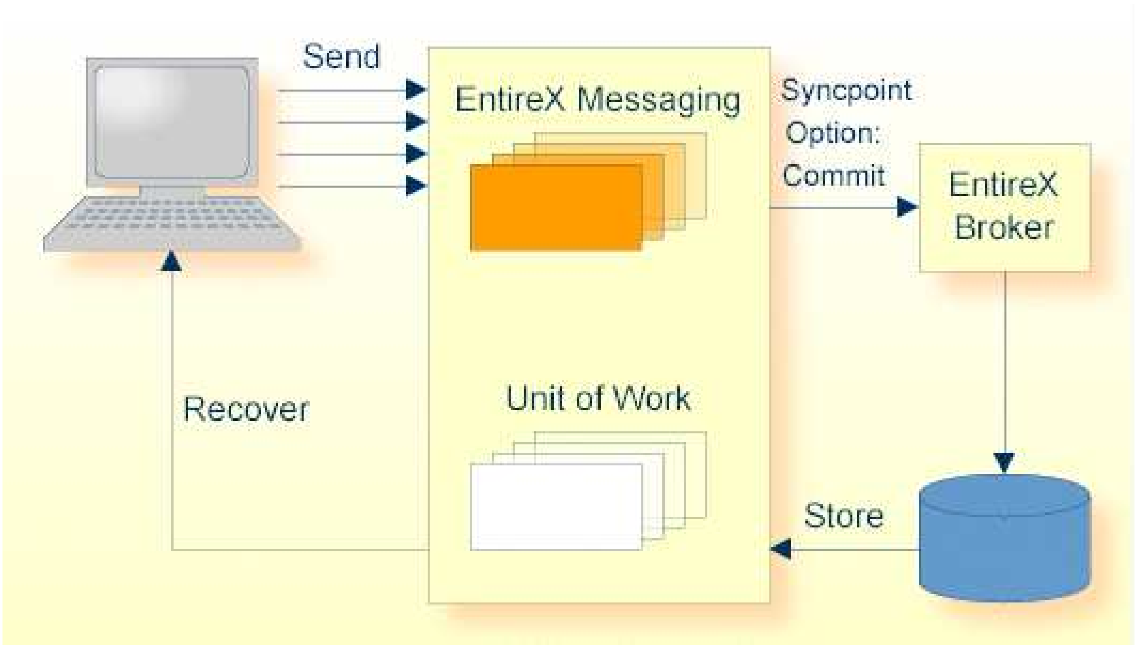
This chapter provides a brief introduction to the concepts of the persistent store and its role in EntireX for providing persistent messaging within the client/server model and also for publish-and-subscribe functionality. It covers the following topics:

The table *Persistent Store Drivers* lists the implementation choices available to each operating system for accessing the physical persistent store. See also *Using Persistence and Units of Work*, *Broker UOW Status Transition* and *Managing the Broker Persistent Store* under z/OS | UNIX | Windows | BS2000/OSD | z/VSE.

Client Server Model: Persistent Messaging

EntireX provides persistent messaging within the client/server model. This is achieved by storing all persistent messages on disk so that if a system failure occurs, messages will automatically be recovered allowing applications to be restarted without any loss of data. The section *Using Persistence and Units of Work* describes implementation issues and how to use persistence and units of work in EntireX Broker. Units of work can also be used without persistence; units of work which are the vehicle for persistent messaging.

The following figure illustrates the concept of persistent messages.



Persistence in an EntireX Broker’s unit of work (a group of logically related messages) has the following four variations:

- Both the unit of work and its status have persistence.
- The unit of work does not have persistence, but its status does.
- The unit of work has persistence, but its status does not.
- Neither the unit of work nor its status has persistence.

The status of a message is information about the message rather than the actual message data itself. This enables the sender to determine the progress of the message and determine if it has been received by the partner and whether processing was successfully completed. This gives applications the option of having the Broker kernel store only the message status and not the message itself, provided the application has been written to resend data from a known point in the event of system failure. This option can afford significant performance benefits over storing the whole message data.

To support transaction control in a coordinated operation of distributed systems, EntireX can group logically related messages into "units of work" that are committed to the EntireX Broker for further transmission when complete. In case of failure on the server side, the receiving program can backout the whole unit of work; this makes it available for processing later or by another server.

Publish-and-Subscribe Model: Persistent Behavior

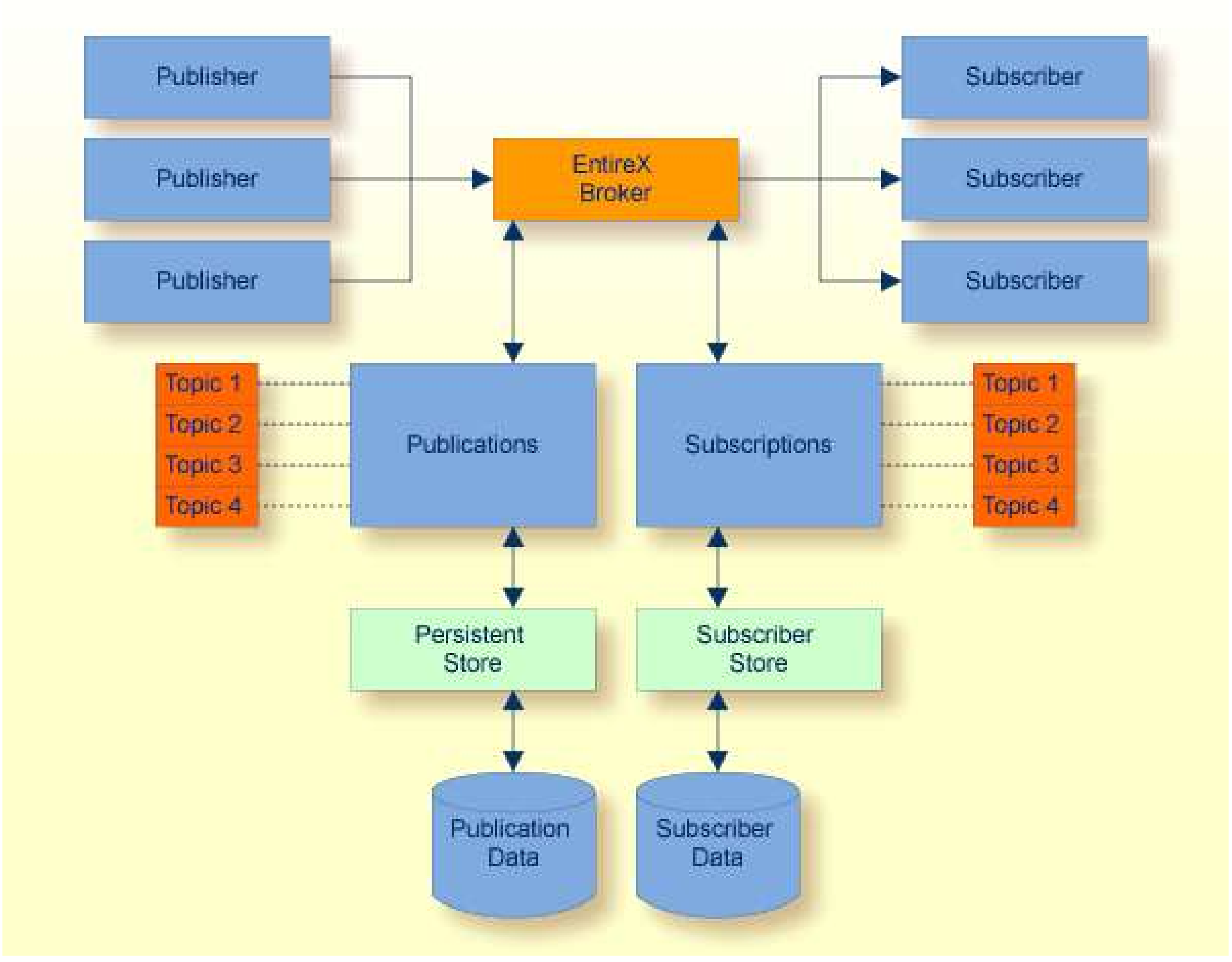
EntireX provides persistent publish-and-subscribe behavior by writing information to disk in order to protect against system failures. This allows applications to be restarted without any loss of the following types of data:

- **Durable Subscription Information**
  This comprises a list of subscribers and the topics to which they have durably subscribed. This ensures that users only have to subscribe once to a topic; their persistent status remains after Broker

is restarted. If the persistent store is used to maintain subscription status, you must define the `SUBSCRIPTION-EXPIRATION` options.

- **Publication Data**
  This data is also persisted if the administrator has defined this characteristic for the topic.

The diagram below shows the two types of publish-and-subscribe information which is written to the persistent store.



Definitions of Persistent Messaging Terms

- UOW
- Persistent Store
- Persistent Store Drivers
- UOW Lifetime
- Persistent UOW
- Persistent Status
- Publication
- Durable Subscription
- Publication Lifetime
- Subscription Expiration

UOW

A unit of work (UOW) is a set of one or more messages that are processed as a single unit. The sender of a UOW adds messages to the UOW and then indicates that the UOW is complete (COMMIT). The UOW and its messages are not visible to the receiver until the sender has committed the UOW. Once the UOW is committed, the receiver can receive the messages, and can indicate when the UOW is complete (COMMIT).

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.

Persistent Store Drivers

A persistent store driver is an executable, or a load module, that implements access to the physical persistent store. There is one persistent store driver for each persistent store type. The following table shows the persistent store options:

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

See also *Managing the Broker Persistent Store* under z/OS | UNIX | Windows | BS2000/OSD | z/VSE and also `PSTORE-TYPE` under *Broker Attributes*.

UOW Lifetime

Each UOW has a lifetime value associated with it. This is the period of time that the UOW is allowed to exist without being completed. This time starts when the UOW is initially created and runs until the UOW is completed. A UOW is completed when it is successfully:

- cancelled or backed out by its sender, or
- cancelled or committed by its receiver.

If the UOW is in ACCEPTED status when this lifetime expires, the UOW is placed into a TIMEOUT status. Lifetime timeouts will not occur when the UOW is in either RECEIVED or DELIVERED status.

A special "pseudo-clock" is maintained for UOW lifetimes. This clock is implemented in such a way that it only runs when the Broker is active. This prevents a UOW lifetime from expiring while the Broker is down or otherwise unavailable.

Persistent UOW

Persistence is an attribute of a UOW (unit of work). If a UOW is persistent, its messages are saved in the persistent store when the sender `COMMIT`s the UOW and they are retained until the receiver `COMMIT`s or `CANCEL`s the UOW, or until its lifetime expires. If the Broker or system should fail after the UOW is committed by the sender, the UOW (and its conversation) will be restored to their last, stable status when the Broker restarts.

Persistent Status

Persistent status is an attribute of a UOW (unit of work). If a UOW has persistent status, the status of the UOW is maintained in the persistent store, and is updated whenever the status changes. The persistent status remains in the persistent store after the UOW is completed, until its status lifetime has expired.

A persistent status value represents a multiple of the UOW lifetime value. Thus if a UOW has a lifetime of 5M (whereby M stands for minutes) and a persistent status value of 4, the status of the UOW would be deleted 20M (5M*4) after the UOW was completed.

Publication

A publication is one or more messages forming an atomic unit and sent by a publisher to a topic. Subscribers are then able to receive publications committed after the time at which a subscriber first subscribes.

Durable Subscription

Subscribers inform EntireX of their intent to receive publications by issuing a `SUBSCRIBE` command and specifying the topic of interest. If the administrator has specified this topic to the Broker attribute file with a characteristic of `DURABLE`, users will be able to subscribe to the topic durably. This means that the user’s subscription status remains after EntireX is restarted.

Publication Lifetime

A characteristic of the topic is the lifetime which publications will live and be available to subscribers. Once a publication has been received by all eligible subscribers, it will be removed automatically, even before its lifetime has been reached.

Subscription Expiration

Subscribers inform EntireX of their intent to receive publications by issuing a `SUBSCRIBE` command and specifying the topic of interest. If the administrator has specified this topic to the Broker attribute file with a characteristic of `DURABLE`, all user subscriptions to that topic will be durable. This means that the user’s subscription status remains after EntireX is restarted.

Availability of Persistent Store

⚠ **Warning:**
The persistent store must be available before you attempt to start or restart the Broker; otherwise your Broker will not initialize.

- Introduction
- Disconnect the Persistent Store
- Connect the Persistent Store

Introduction

The `PSTORE` must be available for the Broker to start. Subsequently, Broker will continue to function even if the `PSTORE` becomes unavailable and applications issuing non-persistent commands will continue without interruption. However, Broker will not be able to process commands relating to persistence until the `PSTORE` becomes available again.

Users issuing commands involving persistence - for example units of work with persistence and durable publish and subscribe - are notified of the unavailability of the `PSTORE` through an ACI return code. In addition, persistent commands being processed at the point of unavailability are backed out, and details of the `PSTORE` problem are written to the Broker log file.

There are several reasons for the `PSTORE` becoming unavailable. Examples:

- unavailability of the `PSTORE` file(s)
- capacity of `PSTORE` file being exceeded
- in the case of Adabas, termination of the database

Disconnect the Persistent Store

You can remove the state "No new Units of Work" - that is, no new persistent data - using CIS. If the `PSTORE` capacity is exceeded, an error message is written to the Broker log file. You must use Command and Information Services (CIS) to ensure that users cannot issue further commands creating new units of work or publications.

During the time the `PSTORE` is unavailable, there is no timeout processing for unit-of-work and publication records kept in the `PSTORE`. In addition, some in-memory resources relating to persistent items, such as conversation control blocks, are also retained until the `PSTORE` becomes available again and normal processing is resumed for all commands.

See executable command request `DISCONNECT-PSTORE` under `ETBCMD`: Executable Command Requests.

Connect the Persistent Store

Subsequently, you can use CIS to make the `PSTORE` available again, allowing users only to issue commands consuming records from the `PSTORE` rather than producing new ones. After a period of operation in this state, the contents of the `PSTORE` will be reduced sufficiently, and you can remove the state "No new Units of Work" through CIS.

See executable command request `CONNECT-PSTORE` under `ETBCMD`: Executable Command Requests.

Migrating the Persistent Store

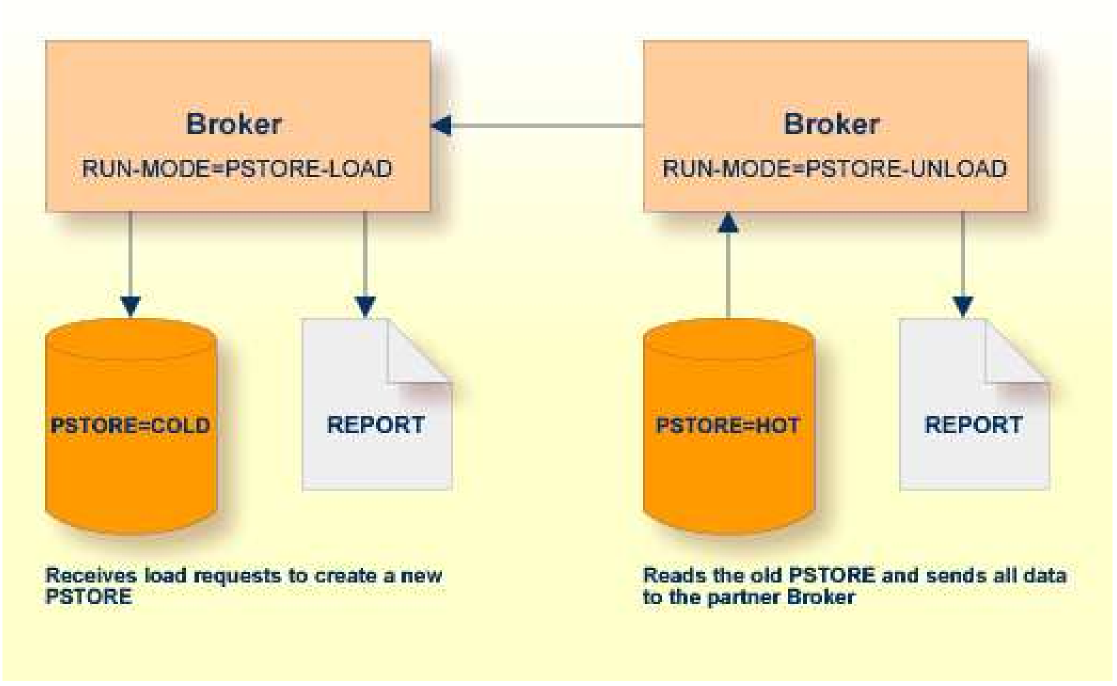
- Introduction
- Configuration
- Migration Procedure

Introduction

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 shall 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.


Broker
RUN-MODE=PSTORE-LOAD
Broker
RUN-MODE=PSTORE-UNLOAD
PSTORE=COLD
REPORT
PSTORE=HOT
REPORT
Receives load requests to create a new PSTORE
Reads the old PSTORE and sends all data to the partner Broker

Configuration

The migration procedure requires two Broker instances, each started with the `RUN-MODE` attribute. 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` under *Broker Attributes* for a 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 persistent 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.

Persistent Store Report

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

- Configuration
- Sample Report

Configuration

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

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

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

The report file is created using the following rules:

BS2000/OSD

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

UNIX

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

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

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

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

Windows

Same as UNIX.

z/OS

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

z/VSE

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

Sample Report

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

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

Identifier        Elements  Total length  Record Type   Date       Time          Action
100000000D000016        5          1148   Conversation  2008-02-21 17:18:57.190  Created
100000000D000017        5          1148   Conversation  2008-02-21 17:18:57.654  Created
100000000D000018        5          1148   Conversation  2008-02-21 17:18:58.122  Created
100000000D000019        5          1148   Conversation  2008-02-21 17:18:58.590  Created
100000000D00001A        5          1148   Conversation  2008-02-21 17:18:59.054  Created
100000000D00001B        5          1148   Conversation  2008-02-21 17:18:59.518  Created
100000000D00001C        5          1148   Conversation  2008-02-21 17:18:59.982  Created
100000000D00001D        5          1148   Conversation  2008-02-21 17:19:00.538  Created
100000000D00001E        5          1148   Conversation  2008-02-21 17:19:01.002  Created
100000000C000001        0             0   Conversation  2008-02-21 17:19:30.676  Deleted
100000000C000002        0             0   Conversation  2008-02-21 17:19:31.675  Deleted
100000000C000003        0             0   Conversation  2008-02-21 17:19:32.675  Deleted
100000000C000004        0             0   Conversation  2008-02-21 17:19:33.675  Deleted
100000000C000005        0             0   Conversation  2008-02-21 17:19:34.675  Deleted
100000000C000006        0             0   Conversation  2008-02-21 17:19:35.675  Deleted
```

The following fields are provided in the report:

- Identifier provides the UOWID (record ID).
- Elements gives the number of messages per UOW when creating or loading records.
- Total Length gives the size of the raw record when creating or loading records.
- Record Type describes the type of the data. Following types are currently supported:
  - Cluster: a special record for synchronization purposes
  - Conversation: a unit of work as part of a conversation
  - Master: a special record to manage the persistent store
  - Publication: a record containing a publication for a durable topic

   - Subscription: a record containing subscriber data (if `SUBSCRIBER-STORE` = `PSTORE`) is defined
- Date and time of the action
- Action describes the action of Broker. The following actions are currently supported:
   - Created: record is created
   - Deleted: record is deleted
   - Loaded: record is loaded (Broker instance with `RUN-MODE` = `PSTORE-LOAD`)
   - Unloaded: record is unloaded (Broker instance with `RUN-MODE` = `PSTORE-UNLOAD`)

Swapping out New Units of Work

The broker processes UOWs in memory. However, if a client produces a large number of UOWs and no server is available, or the server cannot handle all data, the number of UOWs in memory may increase and reach a critical limit.

To avoid an overload of UOWs in memory, EntireX Broker can swap out new conversations that containing UOWs (`STORE=BROKER`) and that have been accomplished by the client with an EOC. The data is persisted on PSTORE and there is no need to keep the data in memory unless a server wants to receive the data.

Activate the swap-out feature with the broker-specific attribute `SWAP-OUT-NEW-UOWS`. It is not activated by default. However, the swap-out feature can be configured per service to define a minimum portion of UOWs kept in memory. Use the service-specific attribute `MIN-UOW-CONVERSATIONS-IN-MEMORY` to define this portion.