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 UNIX, the broker persistent store can be implemented with:

- the Adabas database of Software AG
- the c-tree (C) Copyright database of FairCom Corporation (R)

Note:
If you were previously using the local file system of the machine where the Broker kernel executes, you will need to migrate to a supported persistent store. This persistent store option is no longer supported. To migrate your persistent store, please see the steps outlined in *Migrating the Persistent Store*.

This chapter covers the following topics:

- Implementing an Adabas Database as Persistent Store
- c-tree Database as Persistent Store
- Migrating the Persistent Store

See also *Concepts of Persistent Messaging*.

Implementing an Adabas Database as Persistent Store

- Introduction
- Adabas Persistent Store Parameters
- Configuring and Operating the Adabas Persistent Store
- Adabas DBA Considerations

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, BS2000/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.

Adabas Persistent Store Parameters

Parameters are supplied using the *Adabas-specific Attributes*. See excerpt from the broker attribute file:

```
DEFAULTS=BROKER
  STORE                     = BROKER
  PSTORE-TYPE               = ADABAS
  PSTORE                    = COLD

DEFAULTS=ADABAS
  DBID                      = dbid
  FNR                       = fnr
```

Configuring and Operating the Adabas Persistent Store

Selecting the Adabas Persistent Store Driver

The Adabas Persistent Store driver module is contained within the regular Broker load library or binaries directory. The module adapsi is activated by specifying the PSTORE-TYPE parameter as shown above.

Use the supplied script *persistence.fdu* in the *bin* directory to create a persistent store file in your Adabas database. This script uses the Adabas FDT definition found in file *persistence.fdt* in the *etc* directory.

The script *persistence.fdu* can be executed like this:

```
persistence.fdu <dbid> <fnr>
```

Note:
You can customize the supplied script and FDT file in accordance with your site requirements. See the *Adabas Utilities* manual where necessary, specifically *ADAFDU (File Definition Utility)*.

To run the script file

1. Ensure that you execute the script file on the same machine that the target Adabas is running on. (The database can be either active or inactive at the time you execute it.)
2. Ensure that Adabas environment variables (such as ACLDIR, ADATOOLS, ADABIN and ADALNK) are set up. These environment variables are set by sourcing the corresponding environment scripts. See your Adabas documentation for details.
3. Set your working directory to the one where the fdt file is located.
4. Execute the fdt file, passing it two parameters. (The first one is the DBID, where persistent store file is to be created; the second is the file number.)
5. Option: If the DBID is less than 3 characters long, include leading zeros. For example:

   ```
   persistence.fdu 001 19
   ```

   Result: Creation of file number 19 in database 1.

Defining an Adabas FDT for EntireX File

```
ADACMP FNDEF='01,WK,21,A,DE'
ADACMP FNDEF='01,WJ,126,B,MU'
ADACMP FNDEF='01,WI,126,B,DE,NU'
ADACMP FNDEF='01,WL,96,A,DE,NU'
ADACMP FNDEF='01,WP,96,A,DE,NU'
```

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. Use either the Adabas utility ADAOPR or the Adabas DBA workbench to clean-up any user queue element belonging to the previous Broker job.
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	**Warning:** **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.

Note:
If the Adabas file space is exceeded, Broker will automatically terminate, and it will be necessary either to increase the space available to the file using Adabas utilities or to perform a Broker HOT start with `NEW-UOW-MESSAGES=NO` to allow units of work to be consumed before normal operation can continue.

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, messages records, 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

Calculation Factors	Required Space
• 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
• 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.

Copying the Persistent Store from/to another Adabas File or Database

The DBA can perform an `UNLOAD` of the Adabas file in which the persistent store is located (this must be done when Broker is not running). This allows the persistent store to be LOADed into another Adabas file, in the same or in another Adabas database. Broker can then be restarted (PSTORE=HOT) with the attributes specifying the new location of the persistent store file. See *Table of Adabas Parameter Settings* above. See separate Adabas documentation for details of Adabas utilities for `UNLOAD` and `LOAD` operations.

The persistent store file can only be reloaded into another Adabas database running on the same platform type as the Adabas database from which it was unloaded.

c-tree Database as Persistent Store

EntireX provides a c-tree © persistent driver based on the c-tree© User API of the FairCom Corporation ®. This driver manages a fast and reliable embedded local database.

In order to operate EntireX using the c-tree persistent store option, you must assign Broker attribute `PSTORE-TYPE=CTREE`. No other attributes are required. However, several attributes are supported to set additional optional attributes for the c-tree store. See *c-tree-specific Attributes* under *Broker Attributes* for details.

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
- Migration Procedure

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.