Altibase Tools & Utilities

Utilities Manual

Release 6.1.1

April 23, 2012

ALTIBASE
PERFORMANCE SOLUTIONS

Altibase Tools & Utilities Utilities Manual

Release 6.1.1

Altibase Corporation

10F, Daerung PostTower II, 182-13,

Guro-dong Guro-gu Seoul, 152-847, Korea

Telephone: +82-2-2082-1000   Fax: 82-2-2082-1099

E-mail: support@altibase.com   www: http://www.altibase.com

Contents

Preface

About This Manual

This manual describes how to use the ALTIBASE® HDB™ utilities that are provided for use with ALTIBASE HDB.

Audience

This manual has been prepared for the following ALTIBASE HDB users:

- Database administrators
- Application developers
- Programmers

It is recommended that those reading this manual possess the following background knowledge:

- Basic knowledge in the use of computers, operating systems, and operating system utilities
- Experience in using relational databases and an understanding of database concepts
- Computer programming experience

Software Environment

This manual has been prepared assuming that ALTIBASE HDB 5.5.1 will be used as the database server.

Organization

This manual is organized as follows:

- Chapter1: aexport
- Chapter2: SHMUTIL
- Chapter3: Other Utilities

Documentation Conventions

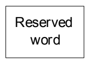
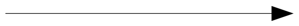
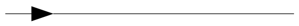
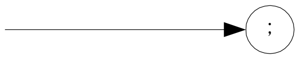
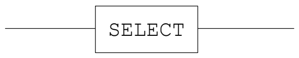
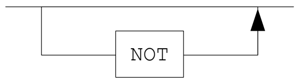
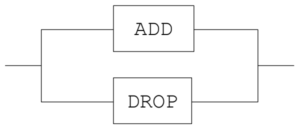
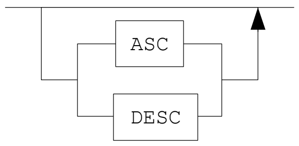
This section describes the conventions used in this manual. Understanding these conventions will make it easier to find information in this manual and in the other manuals in the series.

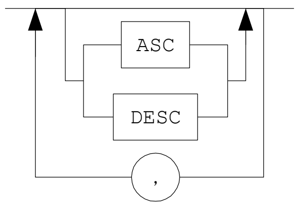
There are two sets of conventions:

- Syntax Diagram Conventions
- Sample Code Conventions

Syntax Diagram Conventions

In this manual, the syntax of commands is described using diagrams composed of the following elements:

Element	Description
	Indicates the start of a command. If a syntactic element starts with an arrow, it is not a complete command.
	Indicates that the command continues to the next line. If a syntactic element ends with this symbol, it is not a complete command.
	Indicates that the command continues from the previous line. If a syntactic element starts with this symbol, it is not a complete command.
	Indicates the end of a statement.
	Indicates a mandatory element.
	Indicates an optional element.
	Indicates a mandatory element comprised of options. One, and only one, option must be specified.
	Indicates an optional element comprised of options.

Element	Description
	Indicates an optional element in which multiple elements may be specified. A comma must precede all but the first element.

Sample Code Conventions

The code examples explain SQL statements, stored procedures, iSQL statements, and other command line syntax.

The following table describes the printing conventions used in the code examples.

Convention	Meaning	Example
[]	Indicates an optional item.	`VARCHAR [(size)] [[FIXED |] VARIABLE]`
{}	Indicates a mandatory field for which one or more items must be selected.	`{ ENABLE | DISABLE | COMPILE }`

|	A delimiter between optional or mandatory arguments.	`{ ENABLE | DISABLE | COMPILE }`
		`[ ENABLE | DISABLE | COMPILE ]`

.
.
.
	Indicates that the previous argument is repeated, or that sample code has been omitted.	`iSQL> select e_lastname from employees;`
		`E_LASTNAME`
		`------------------------`
		`Moon`
		`Davenport`
		`Kobain`
		`.`
		`.`
		`.`
		`20 rows selected.`

Other symbols	Symbols other than those shown above are part of the actual code.	`EXEC :p1 := 1;`
		`acc NUMBER(11,2);`

Italics	Statement elements in italics indicate variables and special values specified by the user.	`SELECT * FROM table_name;`
		`CONNECT` *`userID`*`/`*`password`*`;`

Lower Case Letters	Indicate program elements set by the user, such as table names, column names, file names, etc.	`SELECT e_lastname FROM employees;`

Convention	Meaning	Example
Upper Case Letters	Keywords and all elements provided by the system appear in upper case.	`DESC SYSTEM_.SYS_INDICES_;`

Related Reading

For additional technical information, please refer to the following manuals:

- ALTIBASE HDB Installation Guide
- ALTIBASE HDB Administrator's Manual
- ALTIBASE HDB Replication Manual
- ALTIBASE HDB Precompiler User's Manual
- ALTIBASE HDB ODBC Reference
- ALTIBASE HDB Application Program Interface User's Manual
- ALTIBASE HDB iSQL User's Manual

Online Manuals

Online versions of our manuals (PDF or HTML) are available from the Altibase Download Center (http://atc.altibase.com/).

Altibase Welcomes Your Comments

Please feel free to send us your comments and suggestions regarding this manual. Your comments and suggestions are important to us, and may be used to improve future versions of the manual.

When you send your feedback, please make sure to include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your full name, address, and phone number

Write to us at the following e-mail address: support@altibase.com

For immediate assistance with technical issues, please contact the Altibase Customer Support Center.

We always appreciate your comments and suggestions.

1 aexport

1.1 Overview

1.1.1 Concept

`aexport` is a tool for supporting automated data migration between Altibase databases. `aexport` stores the logical structures and data in text format, and automatically creates a script for use in loading the stored text data into a new ALTIBASE HDB.

The objects and components that `aexport` can extract from a database to which it is connected include the database users, user privileges, tables, tablespaces, table constraints, indexes, views, stored procedures, sequences, and replication objects.

Because `aexport` creates SQL scripts corresponding to the logical structures in the database and downloads all of the data in text form, it can be used to migrate data between different versions of ALTIBASE HDB, or even between different platforms. It is recommended that `aexport` be executed when ALTIBASE HDB is running but not actively providing service, i.e. no when clients are connected.

1.1.2 Features

The database objects and structural elements that `aexport` can extract are as follows:

- Database Users
- User Privileges
- Tablespaces
- Tables
- Table Constraints
- Indexes
- Views
- Stored Procedures
- Replication Objects

When `aexport` is executed, it creates SQL scripts to create the database elements listed above and a shell script to run them.

1.1.3 aexport Modes and Script Files

`aexport` can be run in different modes to extract different portions of the database. The mode in which to run `aexport` is specified on the command line.

The modes in which `aexport` can be run and the SQL script files that are generated in each mode are listed below.

1.1.3.1 Full DB Mode

In this mode, the entire database is extracted. Only the SYS user can execute `aexport` in this mode.

When `aexport` is executed in this mode, the following SQL script files are generated:

- ALL_CRT_DIR.sql: Creates all directory objects
- ALL_CRT_USER.sql: Creates all users
- ALL_CRT_SYNONYM.sql: Creates all synonym objects
- ALL_CRT_REP.sql: Creates all replication objects
- ALL_CRT_VIEW_PROC.sql: Creates all views and stored procedures
- ALL_CRT_TBS.sql: Creates all tablespaces
- ALL_CRT_TBL.sql: Creates all user tables
- ALL_CRT_INDEX.sql: Creates all user indexes
- ALL_CRT_FK.sql: Creates all user-defined foreign keys
- ALL_CRT_TRIG.sql: Creates all user-defined triggers
- ALL_CRT_SEQ.sql: Creates all user-defined sequences
- ALL_CRT_LINK.sql: Creates all user-defined Database Link objects

1.1.3.2 User Mode

This mode is used to export all of the objects owned by the specified user. Only the SYS user or the user whose objects are to be exported can execute `aexport` in this mode. To run `aexport` in user mode, set the -u command-line option to the desired user.

When `aexport` is executed in this mode, the following SQL script files are generated:

- {*User name*}_CRT_TBL.sql: Creates all of the specified user's tables
- {*User name*}_CRT_INDEX.sql: Creates all of the specified user's indexes
- {*User name*}_CRT_FK.sql: Creates all of the specified user's foreign keys
- {*User name*}_CRT_TRIG.sql: Creates all of the specified user's triggers
- {*User name*}_CRT_SEQ.sql: Creates all of the specified user's sequences
- {*User name*}_CRT_LINK.sql: Creates all of the specified user's Database Link objects

1.1.3.3 Object Mode

This mode is used to export the specified set of objects, denoted using the syntax *user.object*. To run `aexport` in object mode, use the -object command-line option.

All of the specified objects must belong to the same user. Additionally, only the SYS user or the user whose objects are to be exported can execute aexport in this mode.

The exception to this rule is that when the SYS user is used to run aexport, any user's objects can be exported.

When aexport is executed in Object Mode, the following SQL script files are generated:

- {*User name*}_{*Object name*}_CRT.sql: Creates the Specified User Object

1.1.3.4 Shell Script Files

In addition to the SQL scripts described above, the following shell script files are created when aexport is executed:

- run_il_in.sh: A data loading script
- run_il_out.sh: A data downloading script
- run_is.sh: A schema creation script
- run_is_con.sh: A constraint creation script. (When the TWO_PHASE_SCRIPT property is set to ON, aexport generates this shell script file. This file includes SQL scripts for creating indexes, foreign keys, triggers and replication objects.)
- run_is_fk.sh: A foreign key creation script. (When the TWO_PHASE_SCRIPT property is set to ON, aexport does not generate this shell script file.)
- run_is_index.sh: An index creation script. (When the TWO_PHASE_SCRIPT property is set to ON, aexport does not generate this shell script file.)
- run_is_repl.sh: A replication object creation script. (When the TWO_PHASE_SCRIPT property is set to ON, aexport does not generate this shell script file.)

When the shell script created by aexport is executed on the destination database, the logical structure of the source database is created on the destination database. Additionally, all of the data that exist on the source database will be loaded into the destination database. The shell script uses iLoader to download and upload the data. However, the process of running iLoader is completely automated within the shell script, so it is not necessary to have a working knowledge of iLoader in order to use aexport.

Because all of the files generated by aexport are text files, the user can modify them as desired, thus enjoying greater flexibility.

Note: Shell script files are not created when aexport is executed in object mode.

1.1.3.5 aexport Properties and Script Files

This section shows which script files are generated depending on the setting of the aexport properties.

Please refer to 1.3 aexport Properties for more information about the properties in the following table.

- When INVALID_SCRIPT = ON, aexport generates INVALID.sql containing SQL scripts for all invalid views and stored procedures, but does not generate any shell script file to execute

INVALID.sql.

- When TWO_PHASE_SCRIPT = ON, `aexport` generates ALL_OBJECT.sql for all objects and ALL_OBJECT_CONSTRAINTS.sql for all indexes, foreign keys, triggers and replication objects. Additionally, the run_is_con.sh shell script file, whose purpose is to execute ALL_OBJECT_CONSTRAINTS.sql, is generated.

1.1.4 Making aexport Settings

`aexport` requires the following information in order to connect to a server:

- ALTIBASE_HOME environment variable

  This is the path where the server or client is installed.

- server_name

  This is the name or IP address of the computer hosting the database from which the data are to be exported.

- port_no

  This is the port number, which is used when connecting via TCP or IPC.

- user_id

  This is the identifier of the database user that `aexport` will use to connect to the database.

- Password

  This is the password corresponding to the user ID described above.

- NLS_USE

  This is the character set in which data are displayed.

The path where the server or client is installed can only be set using the ALTIBASE_HOME environment variable. The rest of the settings can be set using command-line options. For more detailed information on the command-line options, see How to Use aexport.

In order for `aexport` to execute correctly, the ALTIBASE_HOME environment variable must be correctly set, and the `aexport` property settings file (aexport.properties) must exist, and must be suitably configured. For more information about the aexport.properties file, please refer to 1.3 aexport Properties.

Typically, the value of the ALTIBASE_HOME environment variable is automatically set when the server is installed. It is highly recommended that you check to determine that this setting has been properly made. port_no and NLS_USE can be set using either environment variables or the altibase.properties file.

If any of the required information is set in more than one way, the setting that takes precedence is determined as shown below, in descending order of priority:

1. Using command-line options
2. Using environment variables such as ALTIBASE_PORT_NO and ALTIBASE_NLS_USE

3. Using the altibase.properties file.

Therefore, when specifying a required value using a command-line option, it is not necessary to make the appropriate setting using the environment variable or altibase.properties.

If any of the above values are not specified, the user will be prompted to enter the missing values immediately after `aexport` is executed. If the values entered at these prompts are not properly formatted, or are invalid, `aexport` may not work normally.

The exception is the NLS_USE option. The user is not prompted to enter a value for this option, even when it is not set using any of the above methods. Instead, the US7ASCII character set is used by default. This means that if the user does not set the NLS_USE option in an environment in which the US7ASCII character set is not being used, abnormal `aexport` operation and/or data loss can be expected. Therefore, be sure to set the NLS_USE option to a value that is suitable for the operating environment.

In order to ensure that `aexport` operates normally, is recommended that the following environment variables be set:

- ALTIBASE_HOME: This is the path where the server or client is installed.
- ALTIBASE_PORT_NO: This is the port number that is used to connect to the server.
- ALTIBASE_NLS_USE: This is the character set that is used to export and import data.
- PATH: This is the path to the `aexport` executable file. It is typically $ALTIBASE_HOME/bin.

1.1.5 Environment Variables

1.1.5.1 ALTIBASE_HOME

This is used to specify the directory where the package was installed. This must be set in order to use `aexport`.

1.1.5.2 ALTIBASE_PORT_NO

This is the port number with which to connect to the server. In addition to using this environment variable, it can also be specified using the -port command-line option when `aexport` is executed, or can be set in advance in the altibase.properties file.

As noted above, if the settings made using the ALTIBASE_PORT_NO environment variable and the altibase.properties file differ from one another, the environment variable will take precedence. Additionally, specifying the port number using the -port command-line option when running `aexport` will override both settings.

If the port number is not set using any of the above methods, the user will be prompted to enter the value when `aexport` is run.

1.1.5.3 ALTIBASE_NLS_USE

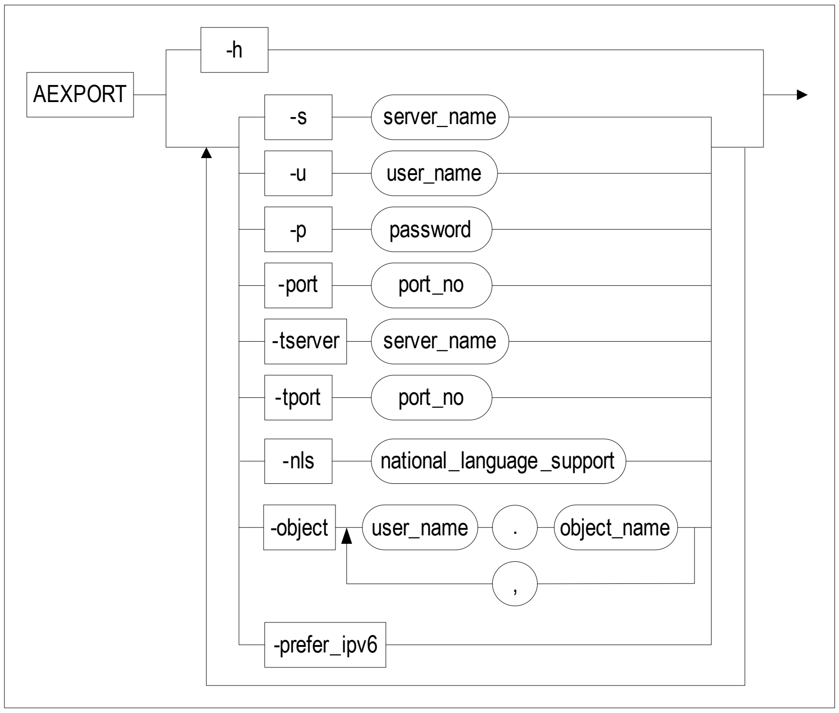
This environment variable is used to specify the character set that is used when connecting to the server. The character set can also be set using the -nls command-line option when `aexport` is executed, or can be set in advance in the altibase.properties file.

As noted above, if the settings made using the ALTIBASE_NLS_USE environment variable and the altibase.properties file differ from one another, the environment variable will take precedence. Additionally, specifying the character set using the -nls command-line option when running `aexport` will override both settings.

Note: If the NLS_USE option is not set using any of the above methods, the US7ASCII character set is used by default. If this setting is not appropriate, `aexport` *may operate abnormally, or data loss may occur. Therefore, be sure to set the NLS_USE option to a value that is suitable for the operating environment.*

1.2 How to Use aexport

1.2.1 Syntax



1.2.2 Parameters

Parameter	Description
-h	Displays the Help menu
-s	This is used to set the host name or IP of the server from which to download data. If this is omitted, an input prompt asking the user to enter the host name will appear. This can be a host name, an IPv4 address, or an IPv6 address. If it is an IPv6 address, it must be enclosed in square brackets (“[ ]“). To specify the localhost, meaning the same computer on which `aexport` is executed, use the computer's host name, the localhost IPv4 address, which is usually 127.0.0.1, or the localhost IPv6 address, which is usually [::1]. For more information about ALTIBASE HDB and IPv6 address notation, please refer to the *ALTIBASE HDB Administrator's Manual*.

Parameter	Description
-u	This is used to set the name of the ALTIBASE HDB user with which to access the server from which the data are downloaded. If this is omitted, an input prompt asking for the user name will appear. In order to perform a Full DB mode export, this option must be set to the SYS user.
-p	This is used to set the password corresponding to the above user. If this is omitted, an input prompt asking for the password will appear.
-object	This is used to set the name(s) of one or more objects to export and the name(s) of the users who own those objects. When this option is specified, `aexport` will run in Object Mode.
-port	This is used to set the port number through which to access the server from which data will be downloaded. If this is omitted, the ALTIBASE_PORT_NO environment variable and the altibase.properties setting will be checked, in descending order of precedence, to determine the port number. If neither of these are set, an input prompt asking the user to enter the port number will appear.
-tserver	This is used to specify the destination server, that is, the server to which the exported data will later be uploaded. This information is written to the script files that are created when `aexport` is executed, and used when those scripts are subsequently executed. As with the -s option, this can be a host name, an IPv4 address, or an IPv6 address.
-tport	This is used to specify the port number through which to access the destination server. This information is written to the script files that are created when `aexport` is executed, and used when those scripts are subsequently executed.
-nls	This is the character set that is used both to export (download) the data from the source database and import (upload) the data to the destination database. At present, the supported character sets are US7ASCII, KO16KSC5601, MS949, BIG5, GB231280, UTF8, SHIFTJIS, and EUCJP.
-prefer_ipv6	This option determines whether to first attempt to resolve a host name to an IPv4 address or an IPv6 address. If a host name is specified for the -s option and this option is used, `aexport` will first attempt to resolve the host name to an IPv6 address. In contrast, if a host name is specified for the -s option and this option is omitted, `aexport` will first attempt to resolve the host name to an IPv4 address. That is, the default behavior is to attempt to resolve the host name to an IPv4 address. If `aexport` fails to connect using the preferred IP address type, it attempts to connect using the other IP address type. For example, when localhost is specified for the -s option and this option is used, `aexport` first tries to connect to the [::1] IPv6 address. If this attempt fails, `aexport` then attempts to connect to the 127.0.0.1 IPv4 address.

1.2.3 The Data Migration Process

The process of using `aexport` to migrate data can be roughly divided into the following steps:

- Generate SQL script files for creating the structure of the objects to be exported from the source database and shell script files for executing the SQL script files
- Export (download) the data from the source database
- Create the required database structures in the destination database
- Import (upload) the data to the destination database
- Create indexes and foreign keys in the target database

1.2.3.1 Exporting the Source Database Structure

`aexport` is first used to generate SQL script files that contain information about the structure of the source database and shell script files for executing the SQL script files.

- Execute `aexport`.

```
$ aexport -s 127.0.0.1 -u sys -p manager
```

- Enter the passwords of the ALTIBASE HDB users at the prompts. This sets the password for each user that is created in the destination database.

  Note that if a password is specified using the USER_PASSWORD `aexport` property, that password is the default password for all users, and this step is skipped.

- When using `aexport` to back up the data on a remote server, indicate the address of the remote server and the port through which to connect to the remote server.

```
$ aexport -s 192.168.1.10 -port 21300 -u sys -p manager
```

1.2.3.2 Exporting Data from the Source Database

Export (download) the data from the source database by executing the shell script that was created by `aexport` in the previous step.

- Check the disk to which the data are to be downloaded to ensure that it has enough free space to hold the data. Because data in text form can occupy more space than the data in internally used data files, it is recommended that the amount of available free space be twice the size of the original data files.
- Execute the “run_il_out.sh” script.

```
$ sh run_il_out.sh
```

1.2.3.3 Creating the Destination Database Structure

Create the required database objects in the destination database.

- Copy all SQL scripts and shell scripts and all 'fmt', 'log', and 'dat' format files created by the “run_il_out.sh” shell script to the system on which the destination database is located. Skip this step if the destination database is on the same system as the source database.

- Start up the destination database.
- Execute the “run_is.sh” script.

```
$ sh run_is.sh
```

- Use iSQL to access the database and check whether all of the required database objects were properly created. If the required database structure was not properly created, inspect the output that was displayed on the screen while run_is.sh was executing to determine the cause of the problem.

1.2.3.4 Importing Data into the Destination Database

Import (upload) the data into the destination database.

- Execute the “run_il_in.sh” script.

```
$ sh run_il_in.sh
```

- Check the directory containing the “run_il_in.sh” shell script file to see whether it contains any files that have the “*.bad” filename extension and are greater than 0 bytes in size. If such a file exists, inspect the contents of the “*.bad” file and the log files related to the table having the same name as the “*.bad” file and take suitable steps to resolve the problem. For more information on how to resolve such problems, please refer to the *iLoader User's Manual*.

1.2.3.5 Creating Indexes and Foreign Keys

Create the required indexes and foreign keys in the target database.

When the TWO_PHASE_SCRIPT property is set to “OFF”:

- Execute the “run_is_index.sh” script.

```
$ sh run_is_index.sh
```

- Execute the “run_is_fk.sh” script.

```
$ sh run_is_fk.sh
```

When TWO_PHASE_SCRIPT=ON:

- Execute the “run_is_con.sh” script.

```
$ sh run_is_con.sh
```

1.2.4 Notes

- When `aexport` is executed with a user account other than that of the SYS user, scripts will be created only for that user's schema.
- When `aexport` is executed with a user account other than that of the SYS user, scripts for replication objects are not created.

- Do not run two or more `aexport` processes at the same time. Because `aexport` uses a temporary table to store created SQL scripts, running two or more `aexport` processes at the same time will yield unpredictable results.
- If the EXECUTE and TWO_PHASE_SCRIPT `aexport` properties are both set to ON and the OPERATION property is set to IN when uploading data, this uploading operation will not be affected by the value of the INDEX property, because no SQL script file dedicated to creating the index is generated.

  Therefore, when it is desired to perform the uploading operation (EXECUTE = ON and OPERATION = IN) with ON for the INDEX property, the TWO_PHASE_SCRIPT property must be set to OFF.
- When the "run_is.sh" script is executed, all existing users and objects will be deleted from the database. Therefore, care must be taken to avoid executing this script on the source database.
- If the -tserver and -tport options are not specified, then the -s and -p parameters are used not only to identify the source server from which data are downloaded, but are also used in all created scripts to identify the destination server to which data will be uploaded.

  To specify a destination server and port that are different from the source server and port, use the -tserver and -tport options. In this case, the -s and -p options will only identify the source server from which data are downloaded, whereas the values specified for the -tserver and -tport options will be written into the created scripts.

  ```
  $ aexport -s 127.0.0.1 -u sys -p manager -tserver 192.168.1.10 -tport 
  21300

  $ cat run_il_in.sh
  iloader -s 192.168.1.10 -port 21300 -u SYS -p MANAGER in -f SYS_T1.fmt -
  d SYS_T1.dat -log SYS_T1.log -bad SYS_T1.bad
  ```

1.2.5 Limitations

1.2.5.1 Creating Stored Procedures using aexport

When a stored procedure is created, if any other stored procedures referenced by that stored procedure do not already exist in the database, the attempt to create the stored procedure will fail.

However, because `aexport` does not have access to information on the interdependencies between stored procedures, there is no guarantee that the stored procedures will be created in the correct order in the destination database. This means that some stored procedure creation attempts may fail.

If this happens, it will be necessary to create those stored procedures manually in the destination database.

1.2.5.2 Creating User-Defined Sequences using aexport

`aexport` only has limited access to meta data pertaining to sequences. Therefore, when `aexport` creates sequences that were originally defined by users other than the SYS user within their own schema, only the value of the INCREMENT BY property will be preserved; the other properties of user-defined sequences (namely, the START WITH, MAXVALUE, MINVALUE, CYCLE, and CACHE properties) will be set to their default values.

If this becomes an issue, it will be necessary to create user-defined sequences manually in the destination database.

1.2.6 Example

1.2.6.1 Execution in Full DB Mode

```
$ aexport -s 127.0.0.1 -u sys -p manager
-----------------------------------------------------------------
     Altibase Export Script Utility.
     Release Version 6.1.1.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
-----------------------------------------------------------------

##### TBS #####

##### USER  #####

##### SYNONYM #####

##### DIRECTORY #####

##### TABLE #####

##### QUEUE #####

##### SEQUENCE ####

##### DATABASE LINK #####

##### VIEW #####

##### STORED PROCEDURE #####

##### TRIGGER #####

##### REPLICATION #####
-------------------------------------------------------
  ##### Follow script files are Generated. #####
  1. run_il_out.sh    : [ iloader formout, data-out script ]
  2. run_is.sh        : [ isql table-schema script ]
  3. run_il_in.sh     : [ iloader data-in script ]
  4. run_is_index.sh  : [ isql table-index script ]
  5. run_is_fk.sh     : [ isql table-foreign key script ]
  6. run_is_repl.sh   : [ isql replication script ]
-------------------------------------------------------

$ ls -l
 ALL_CRT_DIR.sql
 ALL_CRT_FK.sql
 ALL_CRT_INDEX.sql
 ALL_CRT_LINK.sql
 ALL_CRT_REP.sql
 ALL_CRT_SEQ.sql
 ALL_CRT_SYN.sql
 ALL_CRT_TBL.sql
 ALL_CRT_TBS.sql
 ALL_CRT_TRIG.sql
 ALL_CRT_USER.sql
 ALL_CRT_VIEW_PROC.sql
 run_il_in.sh
 run_il_out.sh
 run_is.sh
 run_is_fk.sh
 run_is_index.sh
 run_is_repl.sh
```

1.2.6.2 Execution in User Mode Execution

```
iSQL> CREATE USER user1 IDENTIFIED BY user1;
Create success.

$ aexport -s 127.0.0.1 -u user1 -p user1
-----------------------------------------------------------------
     Altibase Export Script Utility.
     Release Version 6.1.1.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
-----------------------------------------------------------------

##### USER #####

##### SYNONYM #####

##### TABLE #####

##### QUEUE #####

##### SEQUENCE #####

##### DATABASE LINK #####

##### VIEW #####

##### STORED PROCEDURE #####

##### TRIGGER #####
-------------------------------------------------------
  ##### Follow script files are Generated. #####
  1. run_il_out.sh    : [ iloader formout, data-out script ]
  2. run_is.sh        : [ isql table-schema script ]
  3. run_il_in.sh     : [ iloader data-in script ]
  4. run_is_index.sh  : [ isql table-index script ]
  5. run_is_fk.sh     : [ isql table-foreign key script ]
  6. run_is_repl.sh   : [ isql replication script ]
-------------------------------------------------------

$ ls -l
USER1_CRT_DIR.sql
USER1_CRT_FK.sql
USER1_CRT_INDEX.sql
USER1_CRT_LINK.sql
USER1_CRT_SEQ.sql
USER1_CRT_SYN.sql
USER1_CRT_TBL.sql
USER1_CRT_TRIG.sql
USER1_CRT_USER.sql
USER1_CRT_VIEW_PROC.sql
run_il_in.sh
run_il_out.sh
run_is.sh
run_is_fk.sh
run_is_index.sh
```

```
run_is_repl.sh
```

1.2.6.3 Execution in Object Mode

```
iSQL> CREATE USER user1 IDENTIFIED BY user1;
Create success.

iSQL> CONNECT user1/user1;

iSQL> CREATE TABLE t1(i1 INTEGER);
Create success.

iSQL> CREATE VIEW v1 AS SELECT i1 FROM t1;
Create success.

iSQL> CREATE OR REPLACE PROCEDURE proc1(p1 IN INTEGER)
AS a INTEGER;
BEGIN
SELECT * INTO a FROM t1 WHERE i1 = 1;
END;
/
Create success.

$ aexport -s 127.0.0.1 -u user1 -p user1 -object user1.t1
-----------------------------------------------------------------
     Altibase Export Script Utility.
     Release Version 6.1.1.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
-----------------------------------------------------------------

##### TABLE #####

$ ls
user1_t1_CRT.sql

$ aexport -s 127.0.0.1 -u user1 -p user1 -object
user1.t1,user1.v1,user1.proc1
-----------------------------------------------------------------
     Altibase Export Script Utility.
     Release Version 6.1.1.1
     Copyright 2000, ALTIBASE Corporation or its subsidiaries.
     All Rights Reserved.
-----------------------------------------------------------------

##### TABLE #####

##### VIEW #####

##### STORED PROCEDURE #####

$ ls
user1_t1_CRT.sql
user1_v1_CRT.sql
user1_proc1_CRT.sql
```

1.3 aexport Properties

1.3.1 Setting the aexport Properties

Some of the settings that govern the use of `aexport` are made in the aexport.properties file. The aexport.properties file must be located in the $ALTIBASE_HOME/conf directory. (This file is not to be confused with the altibase.properties file, which by default is located in the same directory.)

When ALTIBASE HDB is installed, the $ALTIBASE_HOME/conf directory does not actually contain a file called aexport.properties, but it does contain a sample properties file called aexport.properties.sample. It is thus necessary to copy the aexport.properties.sample, paste it into the same directory, and rename it as aexport.properties before executing `aexport`. If `aexport` cannot find the aexport.properties file in $ALTIBASE_HOME/conf, it will raise an error and terminate.

1.3.2 List of aexport Properties

- OPERATION

  ```
  OPERATION = IN/OUT
  ```

  If this property is set to OUT, scripts for exporting all schemas and data will be created. When the data export script, which consists of iLoader commands, is executed, form files (.fmt) and data files (.dat) will be created.

  If this property is set to IN, the schema creation script and the data loading script, which were created by previously executing `aexport` with this property set to OUT, will be executed, the schema will be created in the destination database, and the data will be loaded into the destination database. The schema creation script and the data loading script can be executed manually at a shell prompt without executing `aexport`.

- EXECUTE

  This property determines whether to automatically execute the scripts that were created.

  ```
  EXECUTE = ON/OFF
  ```

  If it is set to ON, the scripts that are appropriate for the current operation (set using the OPERATION property) will be executed automatically. The file names of these scripts are set using the ILOADER_OUT, ILOADER_IN, ISQL, ISQL_CON, ISQL_INDEX, ISQL_FOREIGN_KEY, and ISQL_REPL properties.

  If it is set to OFF, the scripts will be created, but not executed.

- INVALID_SCRIPT

  This property determines whether to group all of the object creation scripts for invalid objects in a single script file.

  ```
  INVALID_SCRIPT = ON/OFF
  ```

  If this property is set to ON, `aexport` generates a script file named “INVALID.sql”, containing all of the scripts for creating the views and stored procedures that were found to be invalid.

If this property is set to OFF, a SQL script file will be generated for each of the invalid objects in the database; that is, they will be treated just like the valid database objects.

- TWO_PHASE_SCRIPT

  This property determines whether to group all of the object creation scripts in two script files.

  ```
  TWO_PHASE_SCRIPT = ON/OFF
  ```

  If this property is set to ON, `aexport` will create only two SQL script files and two shell script files: ALL_OBJECT.sql, ALL_OBJECT_CONSTRAINTS.sql, run_is.sh, and run_is_con.sh.

  If this property is set to OFF, `aexport` will generate different SQL script files for each of the objects in a database.

- INDEX

  ```
  INDEX = ON/OFF
  ```

  This property determines whether or not to create the indexes when creating the rest of the schema in the destination database. If it is desired to create the indexes after the data have been loaded into the destination database, set this property to ON. It is used when the TWO_PHASE_SCRIPT property is set to OFF.

- USER_PASSWORD

  ```
  USER_PASSWORD = password
  ```

  This property is used to set the password when the users exported from the source database are created in the destination database. (Because `aexport` does not know the passwords of users exported from the source database, the passwords must be manually set.) If this property is not set, a prompt for setting each user's password will appear.

- VIEW_FORCE

  ```
  VIEW_FORCE = ON/OFF
  ```

  If this property is set to ON, views will be forcibly created, even if the underlying tables or other objects don't exist.

- DROP

  This property determines whether to include DROP statements in created scripts.

  ```
  DROP = ON/OFF
  ```

  If this property is set to ON, and if the destination database already contains objects corresponding to those that are to be created, the existing objects will be dropped.

  Because this option specifies that existing objects are to be dropped, it should be used with caution.

  Note: If `aexport` is executed in Object Mode, DROP statements are not generated, regardless of the setting of this property.

- ILOADER_OUT

```
ILOADER_OUT = run_il_out.sh
```

This property determines the name of the shell script file that is created to export (download) the data from the source database. It is used when the OPERATION property is set to OUT.

- ILOADER_IN

```
ILOADER_IN = run_il_in.sh
```

This property determines the name of the shell script file that will be used to import (upload) the data into the destination database.

- ISQL

```
ISQL = run_is.sh
```

This property determines the name of the script file that will be used to create the database schema in the destination database.

- ISQL_CON

```
ISQL_CON = run_is_con.sh
```

This property determines the name of the shell script file that is used to execute the SQL script files for creating indexes, foreign keys, triggers and replication objects. It is used when the TWO_PHASE_SCRIPT property is set to ON.

- ISQL_INDEX

```
ISQL_INDEX = run_is_index.sh
```

This property determines the name of the shell script that will be used to create indexes in the destination database. If no value is specified for this property in the aexport.properties file, this shell script file will not be generated.

- ISQL_FOREIGN_KEY

```
ISQL_FOREIGN_KEY = run_is_fk.sh
```

This property determines the name of the shell script file that is used to execute the SQL script files for creating foreign keys. If no value is specified for this property in the aexport.properties file, this shell script file will not be generated.

- ISQL_REPL

```
ISQL_REPL = run_is_repl.sh
```

This property determines the name of the shell script that will be used to create replication objects in the destination database. If no value is specified for this property in the aex-port.properties file, this shell script file will not be generated.

- ILOADER_FIELD_TERM

```
ILOADER_FIELD_TERM = field_term
```

This property is used to set the field delimiters that are used when the data in tables are saved as text. If this property is not set, the default delimiter between values is the comma (","), no block delimiters are used for numeric values, and double quotation marks (" " ") are used as block delimiters around strings.

Note: The pound (i.e. hash or number sign) character "#" cannot be specified as a delimiter, because it is used to denote comments in the properties file. (The remainder of the line after the "#" will be ignored.)

For more information, please refer to the *ALTIBASE HDB iLoader User's Manual.*

- ILOADER_ROW_TERM

```
ILOADER_ROW_TERM = row_term
```

This property is used to set the delimiter between records that is used when the data in tables are saved as text. If it is not set, the default delimiter is the new line character(s).

Note: The pound (i.e. hash or number sign) character "#" cannot be specified as the record delimiter, because it is used to denote comments in the properties file. (The remainder of the line after the "#" will be ignored.)

For more information, please refer to the *ALTIBASE HDB iLoader User's Manual.*

- ILOADER_PARTITION

This property determines whether or not to create SQL scripts and shell scripts for creating partitions.

```
ILOADER_PARTITION = ON/OFF
```

If this property is set to ON, shell scripts for exporting data from partitions, for creating partitioned tables and all of their partitions, and for importing data into each partition are generated. In other words, enabling this property makes it possible to import data from table partitions in the source database into corresponding partitions in partitioned tables in the destination database.

If this property is set to OFF, whether tables in the source database are partitioned is ignored, and the shell script that is generated creates non-partitioned tables in the destination database and imports data from all partitions of partitioned tables in the source database into corresponding non-partitioned tables in the destination database.

For more information, please refer to the *ALTIBASE HDB iLoader User's Manual.*

1.3 aexport Properties

2 SHMUTIL

2.1 Overview of SHMUTIL

2.1.1 Concepts

SHMUTIL is a utility for checking the status of shared memory that is allocated to ALTIBASE HDB and backing up and deleting databases in this shared memory.

```
shmutil {-p|w|e} [-d home_directory] [-f properties_file_path]
```

2.1.2 Features

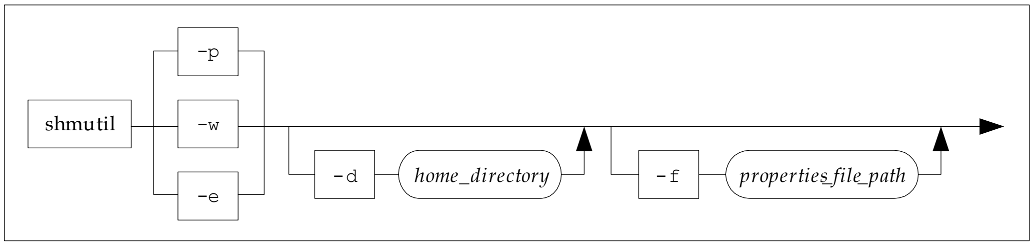
Memory for memory databases is allocated either from the process heap or from shared memory. When shared memory is allocated for storing a memory database, the database remains in the shared memory after ALTIBASE HDB has been shut down. SHMUTIL is therefore provided to check the status of and manage the data stored in shared memory.

SHMUTIL is used to perform the following tasks:

- Checking the status of shared memory
- Backing up a shared memory-resident database to disk
- Removing a database from shared memory

2.2 Using the SHMUTIL Utility

2.2.1 Syntax



2.2.2 Parameters

Parameter	Description
-p	Checks shared memory
-w	Backs up a database loaded into shared memory to disk
-e	Deletes a database from shared memory
-d	Sets the home directory path of ALTIBASE HDB. If this option is not specified, the directory specified by the $ALTIBASE_HOME environment variable is used.
-f	Specifies the location of the ALTIBASE HDB property file. If this option is not specified, the $ALTIBASE_HOME/conf/altibase.properties file is used.

2.2.3 Checking Shared Memory

Displays summarized information about the shared memory that is currently being used to store data, as well as detailed information about individual shared memory segments.

While ALTIBASE HDB is operating normally, the following message will be displayed:

```
$ shmutil -p

ShmUtil: Release 6.1.1.1 - Production on Oct  3 2010 04:52:01
(c) Copyright 2001 ALTIBase Corporation.  All rights reserved.


 #################### IPC Information ######################
   SHM BASE KEY = 2046(0x7fe) ID = 327686(0x50006)

 #################### Brief Information ######################
             SIZE(MB)      PAGE(#)
```

```
    TOTAL     4.0312        129
    USED      3.0000         96
    FREE      1.0312         33
    Timestamp is 1286347518(sec) / 833884(usec)


  #################### Detail Information ######################
      ## Shared Memory Chunk ##
         Shm Key    = 2046
         Shm Id     = 327686
         Page Count = 129
         Size(MB)   = 4.0312
```

For detailed information about how to interpret this output, please refer to the *Administrator's Manual*.

2.2.4 Backing Up Shared Memory

Here is how to use SHMUTIL to back up a database that is loaded into shared memory to disk. ALTIBASE HDB must be completely shut down before the database in shared memory is backed up. In the following example, a database loaded in shared memory is backed up to a directory named *mydbkim*.

```
$ shmutil -w

ShmUtil: Release 6.1.1.1 - Production on Oct  3 2010 04:52:01
(c) Copyright 2001 ALTIBase Corporation. All rights reserved.


 !!!!!! WARNING !!!!!!
 Duplicated DB-Name will overwrite original DB-file.
 Use Different DB-Name.
 Original DB Name 1) => /home2/charlie/work/altidev4/altibase_home/dbs
 Input DB-Path => mydbkim
Tablespace File saving...SYS_TBS_MEM_DIC
Tablespace File saving...SYS_TBS_MEM_DATA
[SUCCESS] Database File Saved
```

2.2.5 Deleting a Database from Shared Memory

Here is how to use SHMUTIL to delete a database that is currently loaded into shared memory. ALTIBASE HDB must be completely shut down before the database is deleted. A database that is resident in shared memory is deleted as follows:

```
$ shmutil -e

ShmUtil: Release 6.1.1.1 - Production on Oct  3 2010 04:52:01
(c) Copyright 2001 ALTIBase Corporation.  All rights reserved.


Ready for Destroying Shared Memory? (y/N)y
Destroyed Shared Memory of Tablespace SYS_TBS_MEM_DIC.
Destroyed Shared Memory of Tablespace SYS_TBS_MEM_DATA.
[SUCCESS] All Shared Memory Removed
[SUCCESS] Database File Saved
```

2.2.6 References

ALTIBASE HDB Administrator's Manual

ALTIBASE HDB Starting User's Manual

3 Other Utilities

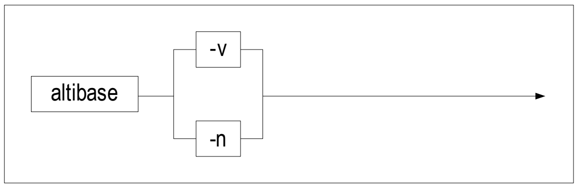
3.1 altibase

3.1.1 About altibase

`altibase` is the executable server file that controls all ALTIBASE HDB services.

```
altibase {-v|n}
```

3.1.2 Syntax



3.1.3 Parameters

Parameter	Description
-v	Displays the version of ALTIBASE HDB that is currently installed
-n	Executes ALTIBASE HDB in the foreground

3.1.4 Description

`altibase` is the executable server file that controls all ALTIBASE HDB services. To start up or shut down ALTIBASE HDB normally in a production environment, do not use this command. Instead, log into iSQL in SYSDBA mode and use the startup or shutdown command, or use the `server` command. The server command is actually a shell comprising several commands related to starting up and shutting down the server. For more information about the `server` utility, please refer to 3.12 server elsewhere in this document. For a complete explanation of how to start up and shut down ALTIBASE HDB, please refer to the *ALTIBASE HDB iSQL User's Manual* or the *ALTIBASE HDB Getting Started*.

When the ALTIBASE HDB server process is started using iSQL, it runs in the background. In contrast, when the `altibase` command is executed at the shell prompt with the `-n` option, ALTIBASE HDB is executed in the foreground. This is used only for ALTIBASE HDB debugging purposes, and is not intended or recommended for live deployment, that is, for use in a production environment. Running the `altibase` executable file with the `-v` option does not start up the server, but merely outputs information about the currently installed version of ALTIBASE HDB.

3.1.5 For More Information

Please refer to the *ALTIBASE HDB Getting Started*, the *ALTIBASE HDB Administrator's Manual*, and the *ALTIBASE HDB iSQL User's Manual* .

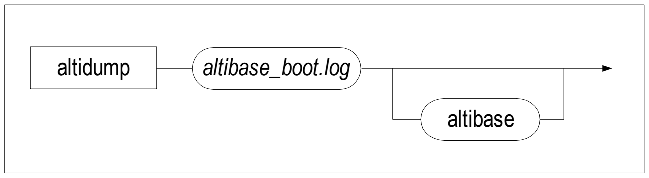
3.2 altidump

3.2.1 About altidump

When ALTIBASE HDB terminates abnormally, information about the ALTIBASE HDB process call stack is recorded in the altibase_boot.log file. `altidump` converts this information into a format that is understandable to the user.

```
altidump altibase_boot.log [altibase]
```

3.2.2 Syntax



3.2.3 Parameters

Parameter	Description
log_file_name (altibase_boot.log)	The log file in which information about the call stack is recorded when ALTIBASE HDB terminates abnormally
altibase	The name and location of the ALTIBASE HDB server binary. If this is not specified, `altidump` will check for the binary in `$ALTIBASE_HOME/bin`.

3.2.4 Description

If, due to abnormal operation of the system or an undiscovered bug, an ALTIBASE HDB server becomes unable to provide service normally, it is necessary to analyze the circumstances at the time of the abnormal termination in order to resolve the issue and prevent future reoccurrences. When ALTIBASE HDB becomes unable to operate normally, it records the process call stack to altibase_boot.log and terminates the server process. Information about the internal ALTIBASE HDB module that was active at the time that ALTIBASE HDB shut down will be stored in the process call stack.

However, this information includes only the subroutines' entries in memory address format within the process, and furthermore is recorded in a format that is not interpretable by users. Therefore, it needs to be converted into a format that can be understood. `altidump` converts the process call stack information recorded in altibase_boot.log into an interpretable format.

In the event of any unexplained abnormal shutdown, use ALTIDUMP to convert the process call stack information in this way and send the resultant data to the Altibase support team. This will enable us to resolve your issue as quickly as possible.

3.2.5 Examples

3.2.5.1 Basic Usage

```
$ altidump $ALTIBASE_HOME/trc/altibase_boot.log
```

3.2.5.2 Other Uses

Because of function overloading in C++, the format of the `altidump` output can still be somewhat indiscernible. `c++filt` can be used to convert this output into a more easily readable form, as shown below:

- SunOS

```
$ altidump $ALTIBASE_HOME/trc/altibase_boot.log | /opt/SUNWspro/bin/
c++filt
```

- Other OS

```
$ altidump $ALTIBASE_HOME/trc/altibase_boot.log | c++filt
```

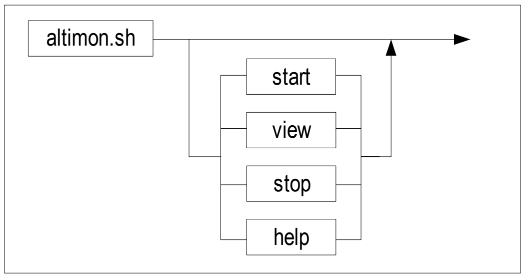
3.3 altimon.sh

3.3.1 About altimon.sh

This utility is used to monitor the status of the ALTIBASE HDB server process.

```
altimon.sh [ start | view | stop | help ]
```

3.3.2 Syntax



3.3.3 Parameters

Parameter	Description
start	Run `altimon` (this option can be omitted)
view	The information that is written to the log file is also echoed to the screen. This process continues until `altimon` is stopped.
stop	Terminates `altimon`
help	Outputs `altimon` help information.

3.3.4 Description

`altimon` continuously monitors the current status of the ALTIBASE HDB server process, threads, and system resources, and records related information to a log file. The log file created by `altimon` provides information that is useful when ensuring the stable operation of the system, and can be used to analyze the cause of any errors that may occur in the system.

`altimon` monitors the following items:

- The ALTIBASE HDB process
- Memory currently in use by ALTIBASE HDB
- Replication

- Sessions
- Transactions that are taking a long time to execute
- The Garbage Collector
- Log files

3.3.5 For More Information

Please refer to the *ALTIBASE HDB Administrator's Manual*.

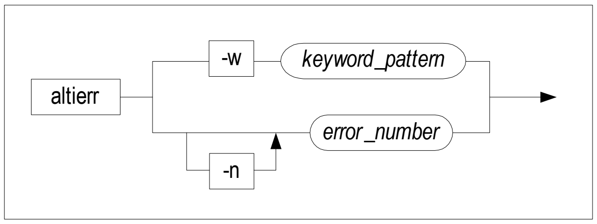
3.4 altierr

3.4.1 About altierr

`altierr` is a utility that is used to search for and display detailed descriptions of ALTIBASE HDB server errors. Errors can be looked up using the error number, or a character string can be used as a search term and sought for within the error messages.

```
altierr {-w keyword_pattern | [-n] error_number}
```

3.4.2 Syntax



3.4.3 Parameters

Parameter	Description
-w	Searches for error message containing the specified search term. All error messages that contain the search term will be displayed.
-n	Searches for an error corresponding to the specified error code number. The error code number can be a hexadecimal number, a positive integer, or a negative integer. Only the record that matches the error code number, if any, will be displayed. When searching for an error using the error code number, the numeric parameter indicator ("-n") can be safely omitted.

3.4.4 Description

The `altierr` utility searches the ALTIBASE HDB errors for strings that contain the specified error message or that match the specified error code number and displays the detailed description of any error that is found. The detailed description of the error includes the error code number, the error code string, the description, the cause of the error, and the action that the user must take in order to remedy the error. When an error occurs, the ALTIBASE HDB server writes the corresponding error code to altibase_boot.log in the following format:

```
ERR-error code
```

`altierr` can be used to search for the detailed description using either a hexadecimal or decimal error code, as shown below:

```
Ex) For 'ERR-00015'
```

```
$ altierr 0x00015
$ altierr -w 00015
$ altierr 21
```

When SQL-related errors occur in applications written using the C/C++ precompiler or applications that use ODBC, the error code will be set in the SQLCODE variable, or will be returned by the ODBC function. In these cases, the error code will be a negative integer. To search for the description of the corresponding error, use `altierr` as follows:

```
Ex) For -266286
```

```
$ altierr -266286
$ altierr 266286
$ altierr 0x4102E
```

`altierr` can be used to search the text of error messages for a search term. In this case, multiple records may be returned. Use a character string as a search term for searching the text of error message descriptions as follows:

```
$ altierr -w connect
$ altierr -w "does not"
```

3.4.5 For More Information

Please refer to the *ALTIBASE HDB Error Message Reference*.

3.5 altipasswd

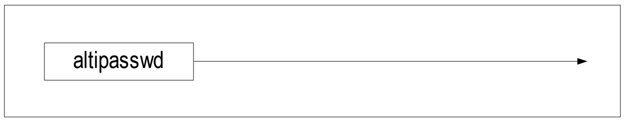
3.5.1 About altipasswd

`altipasswd` is used to changes the password of the SYS user, which is the user provided for accessing the database in SYSDBA mode.

The password of the SYS user must be changed using both this utility and the ALTER USER SQL statement. If the SYS user's password is changed using only the ALTER USER statement, an error will occur when SYSDBA tasks, such as starting up and shutting down the database, are performed.

```
altipasswd
```

3.5.2 Syntax



3.5.3 Description

This utility is used to change the password of the SYS user.

3.5.4 Examples

To change the password of the SYS user from “manager” to “manager1234”, type the following at a shell prompt:

```
$ altipasswd
Previous Password : manager
New Password : manager1234
Retype New Password : manager1234
```

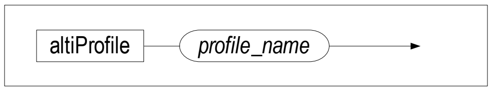
3.6 altiProfile

3.6.1 About altiProfile

If the QUERY_PROF_FLAG property is set to a value other than 0 on an ALTIBASE HDB server, then information about the status of the server and all tasks that are active on the server is recorded to a file for later analysis. Files that contain this information are known as “profiles”. The `altiProfile` utility outputs a profile in text form so that the user can analyze the status of the system.

```
altiProfile {profile_name}
```

3.6.2 Syntax



3.6.3 Description

Converts a server profile to text form.

3.6.4 Examples

```
iSQL> ALTER SYSTEM SET QUERY_PROF_FLAG = 1;
Alter success.

iSQL>   --(Execute an SQL query here.)

$ cd $ALTIBASE_HOME/trc
$ altiProfile alti-1286503704-0.prof
```

3.6.5 How to use altiProfile

The QUERY_PROF_FLAG property must first be set so that the status of the server and the actions that are being performed on the server are logged. Depending on the value of the QUERY_PROF_FLAG property, the following information is logged:

Value	Name	Description
0		No logging.
1	[STATEMENT]	Whenever an SQL statement is executed, the executed SQL statement, execution time, execution information, and information about index and disk access are output.
2	[BIND]	Whenever an SQL statement is executed, BIND parameter(s) is/are output.

Value	Name	Description
4	[PLAN]	Whenever an SQL statement is executed, the execution plan is output.
8	[SESSION STAT]	Every 3 seconds, session information (i.e. the data in V$SESSTAT) is output.
16	[SYSTEM STAT]	Every 3 seconds, system information (i.e. the data in V$SYSSTAT) is output.
32	[MEMORY STAT]	Every 3 seconds, information about memory (i.e. the data in V$MEMSTAT) is output.

The above values can be combined to log the desired information. For example, if the property is set to 1+4+32=37, then whenever an SQL statement is executed, the execution information and execution plan for the SQL statement are output, and additionally, information about memory is output every 3 seconds.

If the QUERY_PROF_FLAG property is set to a nonzero value, the server will create and write to a profile file having a name that follows this convention:

```
alti-#time-#number.prof
```

The `altiprofile` command is used to convert this file into a form that can be analyzed.

3.6.6 Precaution

If the QUERY_PROF_FLAG property is set so as to log information about all SQL statements that are executed on the server, and additionally to log the status of the server and information about all active sessions every 3 seconds, this can place a considerable load on the system. Furthermore, the size of the profile file will increase rapidly, which could cause the disk to become full, consequently causing problems. Therefore, care should be taken when considering whether to perform profiling.

3.6.7 Output

The output format is as follows:

[STATEMENT]

The following table shows the statement-related information that is logged.

Table 3-1 [STATEMENT] Items

Field Name	Value	Description
SQL	String	The SQL statement that was executed
User Info		
User ID	INTEGER	The user identifier
Client PID	BIGINT	The identifier of the client process
Client Type	VARCHAR(40)	The type of the connected client.

Field Name	Value	Description
Client AppInfo	VARCHAR(128)	A string containing information about the client application
Elapsed Time for this SQL statement		
Total	BIGINT	The total query execution time (parsing, validating, optimizing, executing and fetching)
Parse	BIGINT	The time taken to parse the query
Valid	BIGINT	The time taken to validate the query
Optim	BIGINT	The time taken to optimize the query
Execu	BIGINT	The time taken to execute the query
Fetch	BIGINT	The time taken to fetch query results
Query Execution Info		
EXECUTE Result	INTEGER	0: failure 1: rebuild 2: retry 3: queue empty 4: success
Optimizer Mode	BIGINT	The optimization mode
Cost Mode	BIGINT	The optimization cost
Used Memory	BIGINT	Reserved for future use
SUCCESS SUM	BIGINT	The total number of successful executions
FAILURE SUM	BIGINT	The total number of failed executions
PROCESSED ROW	BIGINT	The number of processed records for this SQL statement
Result Set Info		
FETCH Result	INTEGER	0: failure 1: success 2: no results
Index Access Info		
Memory Full Scan Count	BIGINT	The number of full scans that were performed on memory tables
Memory Index Scan Count	BIGINT	The number of index scans that were performed on memory tables
Disk Full Scan Count	BIGINT	The number of full scans that were performed on disk tables

Field Name	Value	Description
Disk Index Scan Count	BIGINT	The number of index scans that were performed on disk tables
Disk Access Info		
READ DATA PAGE	BIGINT	The number of (disk pages that were read from disk for the query
WRITE DATA PAGE	BIGINT	not used
GET DATA PAGE	BIGINT	The number of buffers that were accessed for a disk page during query execution
CREATE DATA PAGE	BIGINT	The number of disk pages that were created during query execution
READ UNDO PAGE	BIGINT	The number of disk pages in UNDO tablespace that were read from disk during query execution
WRITE UNDO PAGE	BIGINT	not used
GET UNDO PAGE	BIGINT	The number of buffers in UNDO tablespace that were accessed for a disk page during query execution
CREATE UNDO PAGE	BIGINT	The number of disk pages in UNDO tablespace that were created during query execution

[BIND]

Outputs information on variables that are bound to the SQL statement

[PLAN]

Outputs the execution plan for the executed SQL statement. For more information on execution plans, please refer to the SQL Tuning chapter in the *Administrator's Manual*.

[SESSION STAT]

Outputs the data in the V$SESSTAT performance view every 3 seconds. For more information on the V$SESSTAT performance view, please refer to the chapter in the *General Reference* that explains performance views.

[SYSTEM STAT]

Outputs the data in the V$SYSSTAT performance view every 3 seconds. For more information on the V$SYSSTAT performance view, please refer to the chapter in the *General Reference* that explains performance views.

[MEMORY STAT]

Outputs the data in the V$MEMSTAT performance view every 3 seconds. For more information on the V$MEMSTAT performance view, please refer to the chapter in the *General Reference* that explains performance views.

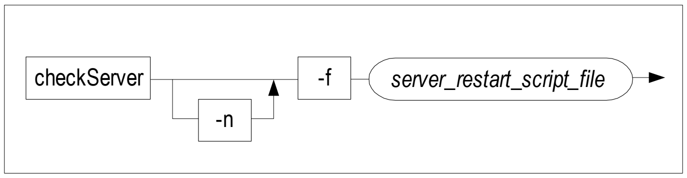
3.7 checkServer

3.7.1 About checkServer

`checkServer` monitors the ALTIBASE HDB process and, if ALTIBASE HDB terminates, executes a script specified by a user.

```
checkServer [-n] {-f server_restart_script_file}
```

3.7.2 Syntax



3.7.3 Parameters

Parameter	Description
-n	Specifies that checkServer is to be executed in the foreground. If this parameter is omitted, `checkServer` will be executed in the background.
-f	The name of the script file to be executed when ALTIBASE HDB terminates

3.7.4 Description

checkServer periodically checks whether the ALTIBASE HDB process is running. If `checkServer` detects that ALTIBASE HDB has terminated, it executes the script specified by a user. It is common to set an ALTIBASE HDB restart script to be executed in the event of termination. Such a restart script can be written as follows:

- The ALTIBASE HDB startup script 'restart.sh'

```
#! /bin/sh
${ALTIBASE_HOME}/bin/server start
```

When `checkServer` is executed, it creates the files *checkServer.pid* and *checkServer.log* in the *$ALTIBASE_HOME/trc* directory. *checkServer.pid* is a kind of lock that prevents another instance of checkServer from being started while the current instance is running. *checkServer.log* is used to regularly record the status of `checkServer`.

If `checkServer` is terminated abnormally, for example by using the command `kill -9`, the *checkServer.pid file* will not be deleted from the *$ALTIBASE_HOME/trc* directory. As long as this file remains in that directory, it will prevent `checkServer` from being executed again.

To terminate `checkServer` normally, use the killCheckServer utility.

Note: `checkServer` executes the specified restart script only when the ALTIBASE HDB server is shut down without using the `server stop` command. When the ALTIBASE HDB server is shut down normally using `server stop`, `checkserver` is also terminated, and thus does not execute the restart script. That is, `checkserver` only considers shutdown using the `server stop` command to be normal shutdown.

3.7.5 Examples

`checkServer` is executed from a shell prompt as follows:

```
$ checkServer -f restart.sh &
```

3.8 dumpla

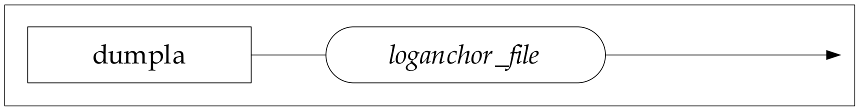
3.8.1 About dumpla

`dumpla` is used to output the contents of loganchor files, which are saved in binary form, in the form of text. Loganchor files contain information that is necessary in order to recover physically stored information (i.e. data files).When a database is created using the CREATE DATABASE statement, ALTIBASE HDB creates these files and stores them with the sequential names loganchor# (where "#" = 0, 1, or 2).

ALTIBASE HDB stores these three files, which have the same contents, in the three respective directories specified using the LOGANCHOR_DIR property in altibase.properties. The reason that three files are maintained is to be prepared in the event that some of the files become lost or corrupt. These files contain information about all of the database's tablespaces and the data files stored in them, as well as recovery-related information. When the database is started, this information is used to load the database into memory and prepare to provide service.

```
dumpla loganchor_file
```

3.8.2 Syntax



3.8.3 Description

Outputs the content of a loganchor file in the form of text.

3.8.4 Examples

At a shell prompt, type the following:

```
$ dumpla loganchor0
```

3.8.5 Output

`dumpla` displays the contents of a loganchor file in the following format:

[LOGANCHOR ATTRIBUTE SIZE]

This section indicates the amount of space that is occupied by each kind of data in the loganchor file. The contents of this section are as follows:

Table 3-2 [LOGANCHOR ATTRIBUTE SIZE] items

Field Name	Value (bytes)	Description
Loganchor Static Area	From 0 (zero) to the maximum value of the unsigned int type	The size of the static information in the loganchor file. Most of this information is information that is required for recovery.
Tablespace Attribute	From 0 (zero) to the maximum value of the unsigned int type	The size of the stored tablespace attributes
Checkpoint Path Attribute	From 0 (zero) to the maximum value of the unsigned int type	The size of the stored checkpoint path attributes
Checkpoint Image Attribute	From 0 (zero) to the maximum value of the unsigned int type	The size of the stored checkpoint image attributes
Disk Datafile Attribute	From 0 (zero) to the maximum value of the unsigned int type	The size of the stored disk datafile attributes

[LOGANCHOR HEADER]

This is loganchor header information, such as the version of the database and the checkpoint Log Sequence Number (LSN). For more information about Log Sequence Numbers, please refer to Table 3-11 DUMPLF Output Items.

The contents of this section are as follows:

Table 3-3 [LOGANCHOR HEADER] items

Field Name	Value	Description
Binary DB Version	Major.minor.patch Ex.) 5.4.1	The version of the database executable file with which the loganchor file was created
Archivelog Mode	Archivelog|No-Archivelog	Indicates whether the database is running in archive mode
Begin Checkpoint LSN	LFGID, FileNo, Offset	The LSN that was current when checkpointing most recently began.
End Checkpoint LSN	LFGID, FileNo, Offset	The LSN that was current when checkpointing was most recently completed.
Disk Redo LSN	LFGID, FileNo, Offset	The redo start point for a DRDB.

Field Name	Value	Description
Global SN	From 0 (zero) to the maximum value of the unsigned long type	Each log has a global Sequence Number (SN) in its header. When a new log is created, the value set for its SN is the SN of the previous log incremented by 1 (previous SN + 1).
SN for Recovery from Replication	From 0 (zero) to the maximum value of the unsigned long type or NULL Refer to Table 3-11 DUMPLF Output Items.	This information is useful when the recovery option, which is an extra feature of replication, has been enabled. It is the sequence number (SN) at which recovery using replication must begin in the event of a fault on an active server.
Server Status	SERVER_SHUTDOWN|SERVER_STARTED	Logs the server state. This value is changed to SERVER_STARTED when the server is started and to SERVER_SHUTDOWN when the server is shut down normally. If this value is already set to SERVER_STARTED when the server starts, this indicates that the server was shut down abnormally, so the server will perform restart recovery.
Log File Group Count	0 - 32	The number of log file groups
Log File Group	0 - 32	Each Log File Group (LFG) comprises the End LSN, ResetLog LSN, Last Created Logfile Num, and Delete Logfile(s) Range fields described below. This LFG ID is used to distinguish them from those in other log file groups.
End LSN	LFGID, FileNo, Offset	The LSN of the first log that is written to when the server is started up after having shut down normally
ResetLog LSN	LFGID, FileNo, Offset	The Reset LSN that was set during incomplete recovery
Last Created Logfile Num	From 0 (zero) to the maximum value of the unsigned int type	The number of the most recently created log file
Delete Logfile(s) Range	Format: [first logfile no. ~ last logfile no.] The numbers of the first and last log files that were deleted.	The range of the most recently deleted log files. When checkpointing is completed, log files that are no longer necessary are deleted. These numbers indicate the range of log files that were deleted.

Field Name	Value	Description
Update And Flush Count	From 0 (zero) to the maximum value of the unsigned int type	The number of times that loganchor files were changed and flushed
New Tablespace ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier for the next new tablespace. When a tablespace is created, this value will be used as its identifier, and will then be incremented.

[TABLESPACE ATTRIBUTE]

This section provides information about the tablespace. The contents of this section are as follows:

Table 3-4 [TABLESPACE ATTRIBUTE] items

Field Name	Value	Description
Tablespace ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the tablespace
Tablespace Name	String Ex.) SYS_TBS_MEM_DIC	The name of the tablespace
New Database File ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier that will be given to the next file to be added to the tablespace
Extent Management	FREE EXTENT BITMAP TABLESPACE	This indicates how extents are managed when a disk tablespace is created. At present, only FREE EXTENT BITMAP TABLESPACE is supported. *Note: If FREE EXTENT BITMAP TABLESPACE is enabled, bitmaps can be used to manage the free extents in a disk tablespace.*
Tablespace Status	Refer to Table 3-5 Possible Tablespace Status Values in [TABLESPACE ATTRIBUTE].	Indicates the current status of the tablespace

Field Name	Value	Description
Tablespace Type	0 - 8 (Refer to Table 3-6 Possible Tablespace Type Values in [TABLESPACE ATTRIBUTE].)	Indicates the type of the tablespace
Checkpoint Path Count	The number of checkpoint paths	The number of checkpoint image file paths. This applies only to memory tablespaces.
Autoextend Mode	AutoExtend|Non-AutoExtend	Indicates whether the tablespace extends in size automatically. This applies only to memory tablespaces
Shared Memory Key	From 0 (zero) to the maximum value of the unsigned int type	The shared memory key for a database that resides in shared memory. This applies only to memory tablespaces
Stable Checkpoint Image Num	0|1	The number corresponding to the set of checkpoint image files that is stable after checkpointing has taken place. This applies only to memory tablespaces
Init Size	From 0 (zero) to the maximum value of the unsigned int type	The initial size (MB) of the tablespace. This applies only to memory tablespaces
Next Size	From 0 (zero) to the maximum value of the unsigned int type	The increment by which the tablespace automatically increases in size (MB). This applies only to memory tablespaces
Maximum Size	From 0 (zero) to the maximum value of the unsigned int type	The maximum size of the tablespace. This applies only to memory tablespaces
Split File Size	From 0 (zero) to the maximum value of the unsigned int type	When a memory tablespace is created, it comprises multiple files of this size. For example, if a tablespace 1 GB in size is to be created and the split file size is 100 MB, then 10 files will be created. This applies only to memory tablespaces

In [TABLESPACE ATTRIBUTE], Tablespace Status can have the following values:

Table 3-5 Possible Tablespace Status Values in [TABLESPACE ATTRIBUTE]

Value	Description
OFFLINE	Currently offline
ONLINE	Currently online
INCONSISTENT	In an inconsistent state
CREATING	Being created
DROPPING	Waiting to be dropped, because the transaction that will drop the database has not been committed yet
DROP_PENDING	The transaction that will drop the database has been committed but the tablespace is still waiting to be dropped because one or more operations are still pending.
DROPPED	Deleted (dropped)
DISCARDED	Discarded
BACKUP	Being backed up
SWITCHING_TO_OFFLINE	Being brought online
SWITCHING_TO_ONLINE	Being taken offline

The possible values of Tablespace Type in [TABLESPACE ATTRIBUTE] are as follows:

Table 3-6 Possible Tablespace Type Values in [TABLESPACE ATTRIBUTE]

Value	Description
0	MEMORY SYSTEM DICTIONARY
1	MEMORY SYSETM DATA
2	MEMORY USER DATA
3	DISK SYSTEM DATA
4	DISK USER DATA
5	DISK SYSTEM TEMP
6	DISK USER TEMP
7	DISK SYSTEM UNDO
8	VOLATILE USER DATA

[MEMORY CHECKPOINT PATH ATTRIBUTE]

This section indicates the path in which checkpoint image files are saved for a memory tablespace. The contents of this section are as follows:

Table 3-7 [MEMORY CHECKPOINT PATH ATTRIBUTE] items

Field Name	Value	Description
Tablespace ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the tablespace
Checkpoint Path	String	The checkpoint image file path

[MEMORY CHECKPOINT IMAGE ATTRIBUTE]

This section indicates the checkpoint image information for a memory tablespace. The contents of this section are as follows:

Table 3-8 [MEMORY CHECKPOINT IMAGE ATTRIBUTE] items

Field Name	Value	Description
Tablespace ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the tablespace
File Number	From 0 (zero) to the maximum value of the unsigned int type	The file number
Create LSN	<LFGID, FileNo, Off-set>	The LSN that was current at the time at which the data file was created.
Create On Disk (PingPong 0)	Created|None	Whether the set of checkpointing files identified by #0 has been created
Create On Disk (PingPong 1)	Created|None	Whether the et of checkpointing files identified by #1 has been created

[DISK DATABASE FILE ATTRIBUTE]

This information indicates the path in which the data file or files for a disk tablespace are saved. The contents of this section are as follows:

Table 3-9 [DISK DATABASE FILE ATTRIBUTE] items

Field Name	Value	Description
Tablespace ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the tablespace

Field Name	Value	Description
Database File ID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the data file
Database File Path	String	The data file path
Create LSN	<LFGID, FileNo, Off-set>	The LSN that was current at the time that the data file was created
Database File Status	Refer to Table 3-10 Database File Status values for [DISK DATABASE FILE ATTRIBUTE	The file state
Atuoextend Mode	AutoExtend| Non-AutoExtend	Whether auto extension mode has been set
Create Mode	0|1	0: The file was reused 1: The file is a newly created file
Initialize Size	From 0 (zero) to the maximum value of the unsigned int type	The initial size (MB) of the data file
Current Size	From 0 (zero) to the maximum value of the unsigned int type	The current size (MB) of the data file
Next Size	From 0 (zero) to the maximum value of the unsigned int type	The increment by which the data file automatically increases in size (MB)
Maximum Size	From 0 (zero) to the maximum value of the unsigned int type	The maximum size (MB) of the data file

In [DISK DATABASE FILE ATTRIBUTE], Database File Status means the following:

Table 3-10 Database File Status values for [DISK DATABASE FILE ATTRIBUTE

Value	Description
OFFLINE	Offline
ONLINE	Online
CREATING	Being created

Value	Description
BACKUP_BEGIN	Backup has started
BACKUP_END	Backup is being completed
DROPPING	Being dropped (deleted)
RESIZING	Being resized
DROPPED	Has been dropped

The following is an example of some of the information output by `dumpla`:

```
[ DISK DATABASE FILE ATTRIBUTE ]
Tablespace ID                  [ 2 ]
Database File ID               [ 0 ]
Database File Path C:\altibase_home\dbs\system001.dbf]
Create LSN                     [ 0, 0, 4443 ]
Database File Status           [ ONLINE ]
Autoextend Mode                [ Non-Autoextend ]
Create Mode                    [ 0 ]
Initialize Size                [10 MBytes(1280 Pages)]
Current Size                   [10 MBytes(1280 Pages)]
Next Size                      [0 MBytes(0 Pages)]
Maximum Size                   [0 MBytes(0 Pages)]
```

3.9 dumplf

3.9.1 About dumplf

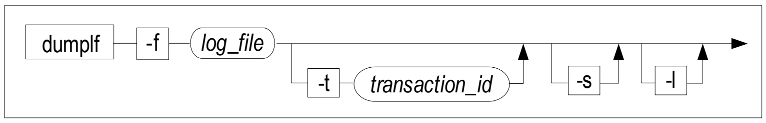
When a transaction performs an operation that changes the contents of the database, such as an INSERT, DELETE or UPDATE operation, changes are made not only to the database's data buffers, but also to log files. These files are maintained for use in performing recovery if it becomes necessary.

To minimize I/O, these logs are recorded in binary format. These log files are stored with the name logfile# (where "#" is the number of the log file, which continuously increments) in the directory specified in the LOG_DIR property in the altibase.properties file.

DUMPLF is a utility that converts and outputs the contents of these log files in text form. These logs can be used to check the type of operations that are performed on the database and determine the frequency of transactions that change the contents of the database.

```
dumplf {-f log_file} [-t transaction_id] [-s] [-l]
```

3.9.2 Syntax



3.9.3 Parameters

Parameter	Description
-f	This is used to specify the name of the file whose contents are to be output. This option must be given. If it is omitted, DUMPLF will terminate and output an error message.
-t	This is used to specify the ID of the transaction for which the logs are to be output.
-s	This option specifies that only the header of the logs is to be output. If this option is omitted, both the header and the body will be output.
-l	Display only information corresponding to log types (LT field) and sub-log-types (OPTYPE and UTYPE fields) in the specified log file.

3.9.4 Description

DUMPLF converts the contents of a log file to text form and outputs it.

3.9.5 Example

At a shell prompt, type the following:

```
$ dumplf -f logfile0
```

3.9.6 Output

The following is an example of DUMPLF output:

```
SN=<10>,LSN=<?,?,820>, COMP:N, MAGIC:820, TID: 6400,BE: N, REP: Y, ISVP: N,
ISVP_DEPTH: 0 PLSN=<0,0,739>, LT: SMR_LT_MEMTRANS_COMMIT, SZ: 45
```

Each field in a log file has the following meaning:

Table 3-11 DUMPLF Output Items

Field Name	Value	Description
SN	From 0 (zero) to the maximum value of the unsigned long type	This is the sequence number that is assigned to each log sequentially
LSN	Format: (LFGID, FileNo, Offset) Offset range: From 0 (zero) to the maximum value of the unsigned int type	This is the log sequence number, which contains information about the physical location of the current log in a log file. The LSN comprises the identifier of the Log File Group (LFG), the file number and an offset value. When DUMPLF cannot determine the LFG and/ or FileNo, question marks ("?") are output. Ex) `(?,?, Offset)`
COMP	Y|N	Indicates whether logs are compressed. Y: Compressed N: Not compressed
MAGIC	From 0 (zero) to the maximum value of the unsigned short type	This value is generated using the log file number and offset portions of the LSN to determine whether a log record is valid. When a Redo or Undo action is performed, even if a log record having garbage data is in a log file, it is possible to determine whether the log record is valid.
TID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the transaction
BE	Y|N	Y: indicates that this log is a Begin Transaction Log, which is recorded when a transaction is started.
REP	Y|N	Y: Indicates that the Sender must check whether to send this log to the Receiver. N: This log is not used by the replication Sender.

Field Name	Value	Description
ISVP	Y|N	"Y" Indicates that this log is an Implicit Save-point Log, that is, the first log that is recorded after the start of execution of a statement. If an error occurs while the statement is executing, the transaction is partially rolled back; that is, it is rolled back only as far as this log.
ISVP_DEPTH	0 - 255	Implicit Savepoint Depth, that is, the nesting depth when a statement is nested within one or more other statements
PLSN	Format: (LFGID, FileNo, Offset) Offset range: From 0 (zero) to the maximum value of the unsigned int type	This value is used to connect all of the logs recorded by the same transaction in a chain.
LT	String Refer to Table 3-12 Possible LT Values in dumplf Output.	Indicates the Log Type (LT).
SZ	From 0 (zero) to the maximum value of the unsigned int type	Indicates the size of the log, in bytes
RdSz	From 0 (zero) to the maximum value of the unsigned int type	Indicates the size of the redo log record, in bytes
DMIOff	From 0 (zero) to the maximum value of the unsigned int type	Indicates the location of the logical log that is used to undo a transaction, or is used for replication
TableOID	From 0 (zero) to the maximum value of the unsigned int type	The object identifier of the table
OID	From 0 (zero) to the maximum value of the unsigned int type	The object identifier of all objects other than tables. This includes record objects.
ContType	0, 1	An internal value that is used for replication
OPTYPE	LogTypeName<LogTypeNumber> Refer to Table 3-13 Possible LogTypeName Values for OPTYPE and UTYPE.	The operation type of a Nested Top Action (NTA) log
AFTER	SZ: <size>, Value: <value>	The after image of the log record

Field Name	Value	Description
BEFORE	SZ: <size>, Value: <value>	The before image of the log record
UTYPE	LogTypeName<Log-TypeNumber> Refer to Table 3-13 Possible Log-TypeName Values for OPTYPE and UTYPE.	The operation type of an UPDATE log
UPOS	Format: (SPA-CEID:<SpaceID>, PID:<PageID>, OFF-SET:<Offset> => OID:<OID>)	The address of the object that was updated. Additionally contains information about what happened during an update operation.
SPACEID	From 0 (zero) to the maximum value of the unsigned short type	The identifier of the tablespace containing the object that was updated
PID	From 0 (zero) to the maximum value of the unsigned int type	The identifier of the page containing the object that was updated
Offset	From 0 (zero) to the maximum value of the unsigned int type	The offset from the beginning of the page con-taining the object that was updated
FLISlot PrevPID NextPID	Format: (<BeforePID> => <AfterPID>)	An internal value used for managing MMDB tablespaces
ESLSN OF LFG	Format: (LFGID, FileNo, Offset) Offset range: From 0 (zero) to the maximum value of the unsigned int type	This is the LSN from which recovery would be performed, if necessary
Lob Locator	From 0 (zero) to the maximum value of the unsigned long type	An internally used value related to the use of replication with the LOB type

The possible values of LT (Log Type) in the DUMPLF output are as follows:

Table 3-12 Possible LT Values in dumplf Output

Value	Description
SMR_LT_DUMMY	Dummy Log
SMR_LT_CHKPT_BEGIN	Checkpoint Begin Log
SMR_LT_DIRTY_PAGE	Dirty Page Log
SMR_LT_CHKPT_END	Checkpoint End Log

Value	Description
SMR_LT_MEMTRANS_COMMIT	Memory Transaction Commit Log
SMR_LT_MEMTRANS_ABORT	Memory Transaction Abort Log
SMR_LT_DSKTRANS_COMMIT	Disk Transaction Commit Log
SMR_LT_DSKTRANS_ABORT	Disk Transaction Abort Log
SMR_LT_SAVEPOINT_SET	Savepoint Set Log
SMR_LT_SAVEPOINT_ABORT	Savepoint Abort Begin Log
SMR_LT_XA_PREPARE	XA Prepare Log
SMR_LT_TRANS_PREABORT	Abort Begin Log
SMR_LT_DDL	DDL (Data Definition Language) Log
SMR_LT_XA_SEGS	XA Prepare Transaction Segment Information
SMR_LT_LOB_FOR_REPL	LOB Log for Replication
SMR_LT_DDL_QUERY_STRING	DDL Query String
SMR_LT_UPDATE	MMDB (Main Memory Database) Update Log
SMR_LT_NTA	MMDB NTA (Nested Top Action) Log
SMR_LT_COMPENSATION	Compensation Log
SMR_LT_DUMMY_COMPENSATION	Dummy Compensation Log
SMR_LT_FILE_BEGIN	File Begin Log
SMR_LT_TBS_UPDATE	Tablespace Update Log
SMR_LT_FILE_END	File End Log
SMR_DLT_REDOONLY	DRDB (Disk-Resident Database) Redo Only Log
SMR_DLT_UNDOABLE	DRDB Undo Log
SMR_DLT_NTA	DRDB NTA Log
SMR_DLT_COMPENSATION	DRDB Compensation Log
SMR_DLT_REF_NTA	DRDB Reference NTA Log
SMR_LT_TABLE_META	Table Meta Log for Replication

Table 3-13 Possible LogTypeName Values for OPTYPE and UTYPE

Value	Description
SMR_OP_SMM_PERS_LIST_ALLOC SMR_OP_SMC_FIXED_SLOT_ALLOC SMR_OP_SMC_VAR_SLOT_ALLOC SMR_OP_SMC_FIXED_SLOT_FREE SMR_OP_SMC_VAR_SLOT_FREE	Logs related to pages and slots in an MMDB

Value	Description
SMR_OP_CREATE_TABLE SMR_OP_CREATE_INDEX SMR_OP_DROP_INDEX SMR_OP_ALTER_TABLE SMR_OP_SMM_CREATE_TBS SMR_OP_INSTANT_AGING_AT_ALTER_TABLE SMR_OP_SMC_TABLEHEADER_ALLOC	Logs related to the execution of DDL statements in an MMDB
SMR_MEM_LOB_CURSOR_OPEN SMR_DISK_LOB_CURSOR_OPEN SMR_LOB_CURSOR_CLOSE SMR_PREPARE4WRITE SMR_FINISH2WRITE	Logs related to controlling LOB values in an MMDB
SDR_OP_SDP_CREATE_TABLE_SEGMENT SDR_OP_SDP_CREATE_LOB_SEGMENT SDR_OP_SDP_CREATE_INDEX_SEGMENT SDR_OP_SDP_ADD_LOB_PAGE_TO_AGINGLIST SDR_OP_SDC_ALLOC_UNDO_PAGE SDR_OP_SDPTB_ALLOCATE_AN_EXTENT_FROM_TBS SDR_OP_SDPTB_ALLOCATE_AN_EXTDIR_FROM_LIST SDR_OP_SDPTB_RESIZE_GG SDR_OP_SDPST_ALLOC_PAGE SDR_OP_SDPSF_ALLOC_PAGE SCT_UPDATE_MRDB_CREATE_TBS SCT_UPDATE_MRDB_CREATE_CIMAGE_FILE SCT_UPDATE_MRDB_DROP_TBS SCT_UPDATE_MRDB_ALTER_AUTOEXTEND SCT_UPDATE_MRDB_ALTER_TBS_ONLINE SCT_UPDATE_MRDB_ALTER_TBS_OFFLINE SCT_UPDATE_DRDB_CREATE_TBS SCT_UPDATE_DRDB_DROP_TBS SCT_UPDATE_DRDB_ALTER_TBS_ONLINE SCT_UPDATE_DRDB_ALTER_TBS_OFFLINE SCT_UPDATE_DRDB_CREATE_DBF SCT_UPDATE_DRDB_DROP_DBF SCT_UPDATE_DRDB_EXTEND_DBF SCT_UPDATE_DRDB_SHRINK_DBF SCT_UPDATE_DRDB_AUTOEXTEND_DBF SCT_UPDATE_DRDB_ALTER_DBF_ONLINE SCT_UPDATE_DRDB_ALTER_DBF_OFFLINE SCT_UPDATE_VRDB_CREATE_TBS SCT_UPDATE_VRDB_DROP_TBS SCT_UPDATE_VRDB_ALTER_AUTOEXTEND SCT_UPDATE_COMMON_ALTER_ATTR_FLAG	Logs related to tablespaces and segments

Value	Description
SDR_OP_SDPST_UPDATE_WMINFO_4DPATH SDR_OP_SDPST_UPDATE_MFNL_4DPATH SDR_OP_SDPST_UPDATE_BMP_4DPATH SDR_OP_SDPSF_ADD_PIDLIST_PVTFREEPIDLIST_4DPATH SDR_OP_SDPSF_MERGE_SEG_4DPATH SDR_OP_SDPSF_UPDATE_HWMINFO_4DPATH SDR_OP_SDP_DPATH_ADD_SEGINFOSET	Logs related to page management for Direct Page Insert in a DRDB
SDR_OP_SDN_INSERT_KEY_WITH_NTA SDR_OP_SDN_DELETE_KEY_WITH_NTA	NTA Logs for DRDB B-tree Indexes
SDR_OP_STNDR_INSERT_KEY_WITH_NTA SDR_OP_STNDR_DELETE_KEY_WITH_NTA	NTA Logs for DRDB R-tree Indexes
SDR_SDP_1BYTE SDR_SDP_2BYTE SDR_SDP_4BYTE SDR_SDP_8BYTE SDR_SDP_BINARY	Physical DRDB logs
SDR_SDP_PAGE_CONSISTENT SDR_SDP_INIT_PHYSICAL_PAGE SDR_SDP_INIT_LOGICAL_HDR SDR_SDP_INIT_SLOT_DIRECTORY SDR_SDP_FREE_SLOT SDR_SDP_FREE_SLOT_FOR_SID SDR_SDP_RESTORE_FREESPACE_CREDIT SDR_SDP_RESET_PAGE SDR_SDP_WRITE_PAGEIMG SDR_SDP_WRITE_DPATH_INS_PAGE	Logs related to page and slot management in a DRDB
SDR_SDPST_INIT_SEGHDR SDR_SDPST_INIT_BMP SDR_SDPST_INIT_LFBMP SDR_SDPST_INIT_EXTDIR SDR_SDPST_ADD_RANGESLOT SDR_SDPST_ADD_SLOTS SDR_SDPST_ADD_EXTDESC SDR_SDPST_ADD_EXT_TO_SEGHDR SDR_SDPST_UPDATE_WM SDR_SDPST_UPDATE_MFNL SDR_SDPST_UPDATE_PBS SDR_SDPST_UPDATE_LFBMP_4DPATH SDR_SDPSC_INIT_SEGHDR SDR_SDPSC_INIT_EXTDIR SDR_SDPSC_ADD_EXTDESC_TO_EXTDIR SDR_SDPTB_INIT_LGHDR_PAGE SDR_SDPTB_ALLOC_IN_LG SDR_SDPTB_FREE_IN_LG	The log related to segment and tablespace management for DRDB.

Value	Description
SDR_SDC_INSERT_ROW_PIECE SDR_SDC_INSERT_ROW_PIECE_FOR_UPDATE SDR_SDC_INSERT_ROW_PIECE_FOR_DELETEUNDO SDR_SDC_UPDATE_ROW_PIECE SDR_SDC_OVERWRITE_ROW_PIECE SDR_SDC_CHANGE_ROW_PIECE_LINK SDR_SDC_DELETE_FIRST_COLUMN_PIECE SDR_SDC_ADD_FIRST_COLUMN_PIECE SDR_SDC_DELETE_ROW_PIECE_FOR_UPDATE SDR_SDC_DELETE_ROW_PIECE SDR_SDC_LOCK_ROW	Logs related to the management of rows in tables in a DRDB
SDR_SDC_UPDATE_LOBDESC SDR_SDC_UPDATE_LOBDESC_KEY SDR_SDC_LOB_WRITE_PIECE SDR_SDC_LOB_WRITE_PIECE4DML SDR_SDC_INIT_LOBPAGE SDR_SDC_LOB_PAGE_TO_AGING_LIST	Logs related to the use of the LOB type in a DRDB
SDR_SDC_PK_LOG	Logs related to the use of primary keys for replication in a DRDB
SDR_SDC_INIT_CTL SDR_SDC_EXTEND_CTL SDR_SDC_BIND_CTS SDR_SDC_UNBIND_CTS SDR_SDC_BIND_ROW SDR_SDC_UNBIND_ROW SDR_SDC_ROW_TIMESTAMPING SDR_SDC_DATA_SELFAGING	Logs related to MVCC for records in a DRDB
SDR_SDC_BIND_TSS SDR_SDC_UNBIND_TSS SDR_SDC_SET_INITSCN_TO_TSS SDR_SDC_INIT_TSS_PAGE SDR_SDC_INIT_UNDO_PAGE SDR_SDC_INSERT_UNDO_REC	Logs related to Transaction Status Slots (TSS) and undo records in a DRDB
SDR_SDN_INSERT_INDEX_KEY SDR_SDN_FREE_INDEX_KEY SDR_SDN_INSERT_UNIQUE_KEY SDR_SDN_INSERT_DUP_KEY SDR_SDN_DELETE_KEY_WITH_NTA SDR_SDN_FREE_KEYS SDR_SDN_COMPACT_INDEX_PAGE	Logs related to B-tree indexes in a DRDB
SDR_SDN_MAKE_CHAINED_KEYS SDR_SDN_MAKE_UNCHAINED_KEYS SDR_SDN_KEY_STAMPING SDR_SDN_INIT_CTL SDR_SDN_EXTEND_CTL SDR_SDN_FREE_CTS	Logs related to MVCC for B-tree index keys in a DRDB

Value	Description
SDR_STNDR_INSERT_INDEX_KEY SDR_STNDR_UPDATE_INDEX_KEY SDR_STNDR_FREE_INDEX_KEY SDR_STNDR_INSERT_KEY SDR_STNDR_DELETE_KEY_WITH_NTA SDR_STNDR_FREE_KEYS SDR_STNDR_COMPACT_INDEX_PAGE	Logs related to MVCC for R-tree index keys in a DRDB
SDR_STNDR_MAKE_CHAINED_KEYS SDR_STNDR_MAKE_UNCHAINED_KEYS SDR_STNDR_KEY_STAMPING	Logs related to R-tree indexes in a DRDB
SMR_PHYSICAL	Physical logs in an MMDB
SMR_SMM_MEMBASE_SET_SYSTEM_SCN SMR_SMM_MEMBASE_ALLOC_PERS_LIST SMR_SMM_MEMBASE_ALLOC_EXPAND_CHUNK SMR_SMM_PERS_UPDATE_LINK SMR_SMM_PERS_UPDATE_NEXT_FREE_PAGE_LINK SMR_SMM_MEMBASE_INFO	Logs related to base information in an MMDB
SMR_SMC_TABLEHEADER_INIT SMR_SMC_TABLEHEADER_UPDATE_INDEX SMR_SMC_TABLEHEADER_UPDATE_COLUMNS SMR_SMC_TABLEHEADER_UPDATE_INFO SMR_SMC_TABLEHEADER_SET_NULLROW SMR_SMC_TABLEHEADER_UPDATE_ALL SMR_SMC_TABLEHEADER_UPDATE_ALLOCINFO SMR_SMC_TABLEHEADER_UPDATE_FLAG SMR_SMC_TABLEHEADER_SET_SEQUENCE SMR_SMC_TABLEHEADER_UPDATE_TABLE_COLUMN_COUNT SMR_SMC_TABLEHEADER_UPDATE_TABLE_SEGMENT SMR_SMC_TABLEHEADER_UPDATE_FLAG_FOR_MEDIA_RECV SMR_SMC_TABLEHEADER_SET_SEGSTOATTR SMR_SMC_TABLEHEADER_SET_INSERTLIMIT SMR_SMC_INDEX_SET_FLAG SMR_SMC_INDEX_SET_SEGATTR SMR_SMC_INDEX_SET_SEGSTOATTR SMR_SMC_INDEX_SET_DROP_FLAG	Logs related to table headers and index headers in an MMDB

Value	Description
SMR_SMC_PERS_INIT_FIXED_PAGE SMR_SMC_PERS_INIT_FIXED_ROW SMR_SMC_PERS_UPDATE_FIXED_ROW SMR_SMC_PERS_UPDATE_FIXED_ROW_NEXT_FREE SMR_SMC_PERS_UPDATE_FIXED_ROW_NEXT_VERSION SMR_SMC_PERS_SET_FIX_ROW_DROP_FLAG SMR_SMC_PERS_SET_FIX_ROW_DELETE_BIT SMR_SMC_PERS_INIT_VAR_PAGE SMR_SMC_PERS_UPDATE_VAR_ROW_HEAD SMR_SMC_PERS_UPDATE_VAR_ROW SMR_SMC_PERS_SET_VAR_ROW_FLAG SMR_SMC_PERS_SET_VAR_ROW_NXT_OID SMR_SMC_PERS_WRITE_LOB_PIECE SMR_SMC_PERS_INSERT_ROW SMR_SMC_PERS_UPDATE_INPLACE_ROW SMR_SMC_PERS_UPDATE_VERSION_ROW SMR_SMC_PERS_DELETE_VERSION_ROW	Logs related to rows in tables in an MMDB

Please refer to the *Atibase Administrator's Manual* for more information about MVCC.

3.10 dumpddf

3.10.1 About dumpddf

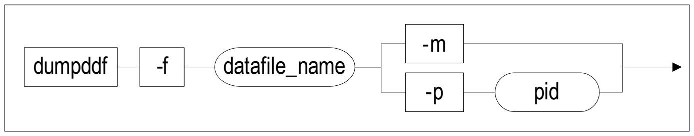
The process of restoring a data file includes the step of reading the header of the data file and/or information about specific pages of the data file.

The `dumpddf` utility checks the header of a data file in order to determine the location within the file at which the recovery process needs to begin.

`dumpddf` also checks information about specific pages in a data file in order to verify the integrity of the data.

```
dumpddf {-f datafile_name} [-m | -p pid]
```

3.10.2 Syntax



3.10.3 Parameters

Parameter	Description
-f	Specifies the name of the data file for which it is desired to obtain information. This option must be given. If it is omitted, DUMDDF will terminate and output an error message.
-m	Outputs the data file header information.
-p	Specifies the ID of the page in the data file for which it is desired to obtain information.

3.10.4 Description

`dumpddf` outputs information about the data file in text form.

3.10.5 Examples

At a shell prompt, type the following:

```
$ dumpddf -f datafile -m

$ dumpddf -f datafile -p page_id
```

3.10.6 Output

The following is an example of `dumpddf` output:

```
[BEGIN DATABASE FILE HEADER]
Binary DB Version              [ 5.4.1 ]
Redo LSN                       [ 0, 0, 734497 ]
Create LSN                     [ 0, 0, 1886 ]
MustRedo LSN                   [ 0, 0, 0 ]
```

In the output, each field has the following meaning:

Table 3-14 Result items for DUMPDDF output

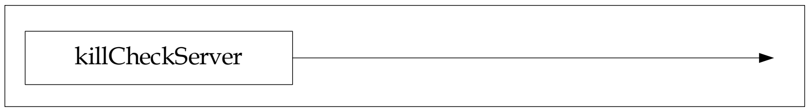
Field Name	Description
Binary DB Version	The version of the data file.
Redo LSN	The redo LSN for media recovery. If the loganchor SN is higher than the Redo LSN of the data file, it will be necessary to perform media recovery, starting from this redo LSN.
Create LSN	The LSN at the time when the specified datafile was created.
MustRedo LSN	Indicates that recovery must be performed up to this redo LSN.

3.11 killCheckServer

3.11.1 About killCheckServer

killCheckServer terminates the checkServer utility if it is currently running.

3.11.2 Syntax



3.11.3 Description

killCheckServer terminates the checkServer utility if it is currently running.

3.11.4 Examples

At a shell prompt, enter the following:

```
$ killCheckServer
```

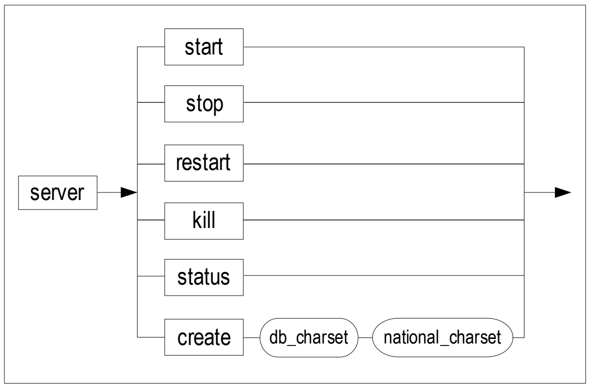
3.12 server

3.12.1 Overview

`server` is a shell script that is used to create, start up, shut down and check the status of an ALTIBASE HDB.

```
server { start | stop | restart | kill | status |
         create db_charset national_charset }
```

3.12.2 Syntax



3.12.3 Parameters

Parameter	Description
start	Starts up the ALTIBASE HDB process
stop	Shuts down the ALTIBASE HDB process
restart	Restarts the ALTIBASE HDB process
kill	Forcibly terminates the ALTIBASE HDB process
status	Displays the status of the ALTIBASE HDB process
create	Creates a database that is 10MB in size, runs in noarchivelog mode, and uses the specified character sets

3.12.4 Description

Typically, iSQL is used to execute SQL statements for creating, starting up and shutting down an ALTIBASE HDB. These frequently used commands have been combined and provided in the form of the `server` shell script for the convenience of DBAs.

The `server` script includes the following functionality:

- Starting up the ALTIBASE HDB process
- Shutting down the ALTIBASE HDB process
- Restarting the ALTIBASE HDB process
- Forcibly terminating the ALTIBASE HDB process
- Displaying the result of querying “select * from tab;”
- Creating an ALTIBASE HDB

For more information about using SQL to manage Altibase databases, please refer to the *ALTIBASE HDB iSQL User's Manual*.

3.12.5 Example

The `server` shell command is used as follows:

```
$ server start
$ server restart
$ server stop
$ server status
$ server kill
$ server create ksc5601 utf16
```

3.12.6 For More Information

Please refer to the *ALTIBASE HDB Administrator’s Manual* and the *ALTIBASE HDB iSQL User’s Manual*.

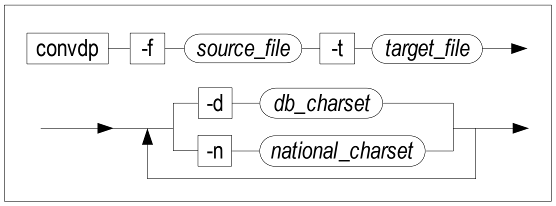
3.13 convdp

3.13.1 Overview

The `convdp` utility converts data from the DataPort file (.dpf) character set to the character set of the destination database.

```
convdp {-f source_file} {-t target_file}
       {[-d db_charset] [-n national_charset]}
```

3.13.2 Syntax



3.13.3 Parameters

Parameter	Description
-f	The name of the file containing the data to be converted
-t	The name of the file to which the converted data are to be written
-d	The database character set into which data having the CHAR and VARCHAR datatypes will be converted
-n	The national character set into which data having the NCHAR and NVARCHAR datatypes will be converted

3.13.4 Description

When the character set and/or the national character data of the data in a DataPort file (.dpf) differ from those of the destination database, the `convdp` utility can be used to convert the data in the DataPort file to the character set and/or the national character set of the target database.

A DataPort file contains header information, the definition of the source table or tables, and the actual data (records) in the table(s). The `convdp` utility converts data having the CHAR and VARCHAR data types to the specified character set, converts data having the NCHAR and NVARCHAR data types to the specified national character set, and writes the converted data to the specified target file.

When searching for the file on which to perform the conversion and determining the name of the output file, `convdp` follows the DataPort naming conventions specified in the chapter of the *Stored Procedures Manual* in which DataPort is explained.

3.13.5 Example

`convdp` is used as follows:

```
iSQL> CREATE TABLE t1 (i1 INTEGER, i2 VARCHAR(100));
Create success.

iSQL> INSERT INTO t1 VALUES (1,'abc');
1 row inserted.

iSQL> EXEC EXPORT_TO_FILE ('SYS','T1','test');
Export - SYS_T1 1 record(s).
Execute success.

$ cd $ALTIBASE_HOME/dbs

% convdp -f test_0 -d 'UTF8' -t test_utf8

ConvDP: Release 5.5.1.0 - Production on Oct 28 2010 04:46:58
(c) Copyright 2001 ALTIBase Corporation.  All rights reserved.

Convert Charset:
[0  ]INTEGER
[1  ]VARCHAR         (KSC5601->UTF8)

Convert 1 rows...
Done.
```

3.13.6 For More Information

Please refer to the *ALTIBASE HDB Stored Procedures Manual*.

3.13 convdp

Index