

HCL Informix 14.10

Migrating and upgrading



Contents

Chapter 1. Migrating and upgrading.....	1
Migrating Informix® database systems.....	1
Overview of migration.....	1
Migration to and reversion from 14.10.....	6
Migration of data between database servers.....	62
Data migration utilities.....	64
Index.....	133

Chapter 1. Migrating and upgrading

You can upgrade to the 14.10 release of HCL Informix® or migrate from other database servers to Informix®. Upgrading is an in-place migration method that uses your existing hardware and operating system software. Some changes to the Informix® database server can affect upgrading from a previous release.

Upgrade tasks

To upgrade to Informix® version 14.10:

1. Read about important migration information, known and fixed customer-reported defects, and platform-specific actions that you must take to configure and use HCL Informix® products: Release, documentation, and machine notes for HCL Informix®
2. Understand your migration path and plan your migration: [Overview of Informix® migration](#)
3. Prepare for migration, including reviewing changes to Informix® products since the release from which you are migrating: [Preparing for migration on page 7](#)
4. Do the migration tasks that are appropriate to your system: [Migrating to the new version of Informix® on page 40](#)
5. Finish the migration process: [Completing required post-migration tasks on page 46](#)

If you for any reason you must revert to your previous version of Informix®, see [Reverting from Informix® Version 14.10 on page 51](#).

Migrating Informix® database systems

The *Informix® Migration Guide* describes how to move data manually between databases, servers, and computers.

These topics are intended for database server administrators or database administrators who are responsible for upgrading the database server or migrating data. These topics assume that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience with database server administration, operating-system administration, or network administration

Overview of migration

Overview of moving data

If you are installing the new version of the database server on another computer or operating system (non-in-place migration), you can use one of several tools and utilities to move data from your current database server.

For example, suppose you migrated to the current version of Informix® and created a few new databases, but decide to revert to the previous version. Before you revert, you can use one of the data-migration tools to save the data you added. After reverting, you can reload the data.

Before you move data, consider these issues:

- Changes in the configuration parameters and environment variables
- Amount of memory and dbospace space that is required
- Organization of the data
- Whether you want to change the database schema to accommodate more information, to provide for growth, or to enhance performance

For information about how to move data between database servers on different operating systems, also see [Migrating database servers to a new operating system on page 62](#).

For information about how to move to a different GLS locale, see the *Informix® GLS User's Guide*.

Prerequisites before moving data

Before you use any data migration utility, you must set your PATH, INFORMIXDIR, and INFORMIXSERVER environment variables.

For information about environment variables, see the *Informix® Guide to SQL: Reference*.

Data-migration tools

Informix® provides tools, utilities, and SQL statements that you can use to move data from one HCL Informix® database to another or from one operating system to another.

You might want to use a data-migration tool when you have different page sizes or code pages. For example, UNIX™ or Linux and Windows store data in different page sizes.

When your migration involves migrating between different operating systems, you must export data and its schema information from one database server and import the exported data into the other database server.

Normally, if you are migrating on the same operating system, you do not need to load and unload data.

You can use the following tools to move data:

- The **dbexport** and **dbimport** utilities
- The **dbload** utility
- The **onunload** and **onload** utilities
- UNLOAD and LOAD statements
- Nonlogging raw tables

When you import data from non-Informix® sources, you can use the following tools:

- The **dbimport** and **dbload** utilities
- External tables that you create with the CREATE EXTERNAL TABLE statement

The best method for moving data depends on your operating system and whether you want to move an entire database, selected tables, or selected columns from a table. The following table summarizes the characteristics of the methods for

loading data and the advantages and disadvantages of each method. The table also shows the database servers on which you can use the tools.



Note: **dbimport** cannot be used to move datablade data between Informix versions.

Table 1. Comparison of tools for moving data

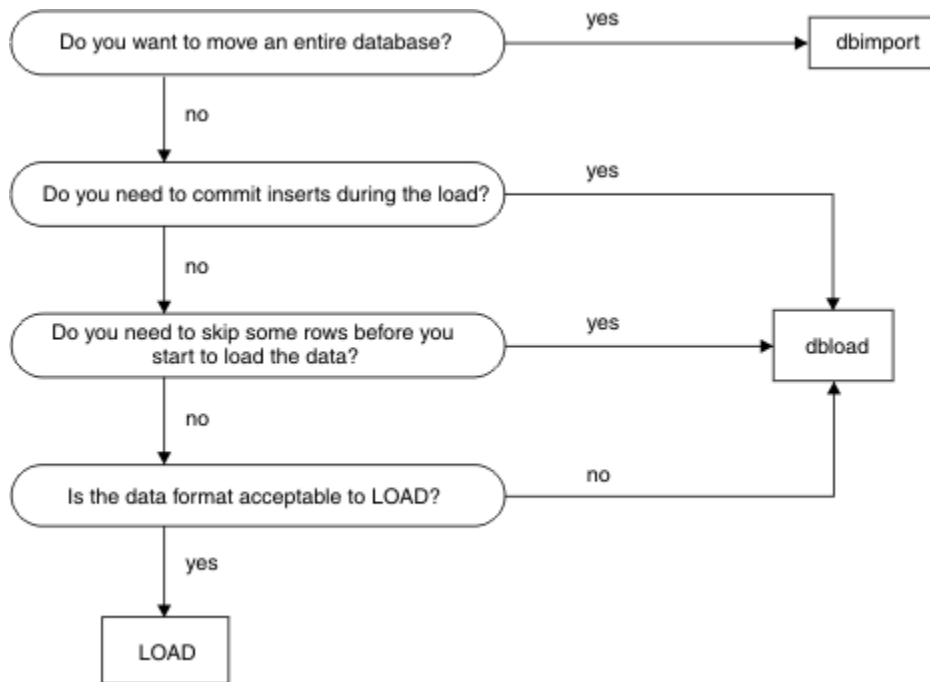
Tool	Description	Advantages	Disadvantages
dbexport and dbimport utility	Imports or exports a database to a text file that is stored on disk or tape	Can modify the database schema and change the data format Can move data between operating systems Optional logging Can import data from non-Informix® sources	Faster performance than the dbload utility, but slower performance than the onload utility Moves the entire database
dbload utility	Transfers data from one or more text files into one or more existing tables	Can modify database schema Can move data between operating systems Optional logging Moderately easy to use Can import data from non-Informix® sources	Slower performance than the dbexport , dbimport , and onload utilities
onunload and onload utilities	Unloads data from a database into a file on tape or disk; loads data, which was created with the onunload command, into the database server	Fast performance Optional logging	Only moves data between database servers of the same version on the same operating system Cannot modify the database schema Logging must be turned off Difficult to use
UNLOAD and LOAD statements	Unloads and loads specified rows	Can modify database schema Can move data between operating systems Easy to use Optional logging	Only accepts specified data formats

Table 1. Comparison of tools for moving data (continued)

Tool	Description	Advantages	Disadvantages
Nonlogging raw tables	Loads certain kinds of large tables	Can load very large data warehousing tables quickly	Does not support primary constraints, unique constraints, and rollback Requires SQL Not recommended for use within a transaction
External tables	Enables you to read and write from a source that is external to the database server, providing an SQL interface to data in text files managed by the operating system or to data from a FIFO device.	Performs express (high-speed) and deluxe (data-checking) transfers	Requires SQL

If you are choosing a tool for loading data, the questions shown in [Figure 1: Choosing among dbimport, dbload, and LOAD on page 4](#) will help you make a decision.

Figure 1. Choosing among dbimport, dbload, and LOAD



In addition to the tools that move data, you can use the **dbschema** utility, which gets the schema of a database and redirects the output to a file, so you can provide the file to DB-Access to re-create the database.

Related information

Migrating Informix (non-in-place migration)

Migration tools

[The dbexport and dbimport utilities on page 65](#)

[The onunload and onload utilities on page 117](#)

[The dbload utility on page 82](#)

[The dbschema utility on page 98](#)

[The LOAD and UNLOAD statements on page 115](#)

When TEXT and BYTE data is scanned, not compressed

The Informix® database server scans TEXT and BYTE data into an existing table when you load data by using the SQL LOAD statement, the **dbload** utility, the Informix® ESQL/C program, or external tables.

Informix® database servers do not have any mechanisms for compressing TEXT and BYTE data after the data has been scanned into a database.

Moving data between computers and dbspaces

You can move data between different computers, and you can import data from environments other than the Informix® database server. Except when you use external tables, you must unload your data to ASCII files before you move the data to another computer.

If you are moving data into the Informix® database server on another computer, you can use the **dbimport** and **dbload** utilities to load the data that you exported.

If you are moving data to an application that is not based on Informix®, you might need to use the UNLOAD statement because you can specify the delimiter that is used in the data files.

Importing data from a non-Informix® source

The **dbimport** and **dbload** utilities can import data from any ASCII file that is properly formatted.

Most applications that produce data can export the data into files that have a suitable format for **dbimport**. If the format of the data is not suitable, use UNIX™, Linux™, or Windows™ utilities to reformat the data before you import it.

In addition to **dbimport** and **dbload**, HPL provides ways to access information from non-Informix® sources.

Moving data by using distributed SQL

If you want to move data with different binary pages and page sizes across platforms and you have expertise in using distributed SQL, you can use INSERT and SELECT SQL statements to transfer the data.



Important: Do not use INSERT and SELECT statements to move data if the database contains BLOB data types.

Prerequisites: A network connection must exist between database server instances.

To move data using INSERT and SELECT statements with fully qualified table names:

1. Capture the complete database schema from the source database server.
2. Alter the extent sizing and, if necessary, the lock modes on tables from page to row.
3. Create and verify the schema on the target database server.
4. Disable logging on both source and target servers where necessary.
5. Create and run the following scripts:
 - a. Create and run separate scripts for:
 - Disabling select triggers on the source server
 - Disabling indexes, triggers and constraints for each table on the target database server.
 - b. Create and run one script per table for the fully-qualified INSERT and SELECT statements.

For example:

```
INSERT INTO dbname@target:owner.table SELECT *  
FROM dbname@source:owner.table
```

You can run the scripts in parallel. In addition, for larger tables, you can create multiple scripts that can partition the table to run in parallel.

- c. Create and run separate scripts for enabling indexes, triggers and constraints for each table
6. Run UPDATE STATISTICS on system catalog tables and stored procedures and functions on the target database server.
7. Adjust starting values for all tables that have serial columns on the target database server.
8. Turn on transaction logging on the source and target database servers.
9. Return the source and target database servers to multi-user mode.
10. Validate the data that was transferred to the target database server.

For information about INSERT and SELECT statements, refer to the *Informix® Guide to SQL: Syntax*. For information on distributed transactions, refer to the *Informix® Administrator's Guide* and the *Informix® Administrator's Reference*.

Related information

Paths for migration to the new version

Migration to and reversion from 14.10

Preparing for migration to Version 14.10

Before you install the new version of Informix®, you must prepare the database server environment for migration by performing specified pre-migration tasks. If you are also migrating from 32-bit to 64-bit database servers, you must perform additional tasks.

Related information

[Upgrading Informix \(in-place migration\)](#)

[Types of migration](#)

[Migrating Informix \(non-in-place migration\)](#)

Preparing for migration

Preparing for migration includes gathering information about and backing up your data, so that you can reinstall the previous version of the server and restore your data if you have a migration problem. Preparing for migration is crucial for successful migration.

Before you begin

- Check the [Support Portal](#) for the latest patches that you must install before you migrate or upgrade Informix® software.
- If you use Enterprise Replication, you must first prepare your replication environment for migration. For more information, see [Enterprise Replication and migration on page 17](#).

About this task

Review and complete all tasks that apply:

1. [Reviewing changes in Informix product functionality on page 8](#).
2. [Checking and configuring available space on page 8](#).
3. [Configuring for recovery of restore point data in case an upgrade fails on page 10](#).
4. Renaming user-defined routines (UDRs) that have the following names: CHARINDEX(), LEFT(), RIGHT(), INSTR(), DEGREES(), RADIANS(), REVERSE(), SUBSTRING_INDEX(), LEN(), and SPACE().
These names are reserved for built-in SQL string manipulation functions.
5. Adjusting settings:
 - a. If you use UNICODE, ensure that the GL_USEGLU environment variable on the source server is set to the same value as the GL_USEGLU environment variable on the target server.
 - b. If the source version of the database server contains the IFX_EXTEND_ROLE configuration parameter, which controls authorization to register DataBlade® modules or external UDRs, disable the parameter by setting it to `0` (off).
6. [Saving copies of the current configuration files on page 11](#).
7. [Preparing 12.10 BSON columns with DATE fields for upgrade on page 12](#).

8. [Closing all transactions and shutting down the source database server on page 13.](#)
9. [Initiating fast recovery to verify that no open transactions exist on page 13.](#)
10. [Verifying the integrity of the data on page 14.](#)
11. [Verifying that the database server is in quiescent mode on page 14.](#)
12. [Making a final backup of the source database server on page 15.](#)



Important: Complete the previous step in case you have to revert to the source database server.

13. [Verifying that the source database server is offline on page 15.](#)

What to do next

If you use high-availability clusters, you must complete additional preparations. See [High-availability cluster migration on page 21.](#)

Related information

[Pre-migration checklist of diagnostic information on page 15](#)

[Hardware and operating system requirements](#)

Reviewing changes in Informix® product functionality

Changes to Informix® product functionality might affect your plans for migrating to the latest version of the product.

About this task

Changes in functionality in Informix® 14.10 can potentially impact your applications, scripts, maintenance processes, and other aspects that are related to your database server environment.

Changes to functionality that was introduced before Informix® 14.10 can also affect your plans.

Checking and configuring available space

Before you migrate to the new version of Informix®, you must make sure that you have enough available space for the new server, your data, and any other network and data tools that you use.

During migration, Informix® drops and then recreates the **sysmaster** database. Depending on which version of Informix® you migrate from, the **sysmaster** database in the current version can be significantly larger.

When you migrate to Version 14.10, you need the following space for building **sysmaster**, **sysutils**, and **sysadmin** databases:

- 21892 KB of logical-log spaces (or 10946 pages) for 2 K page platforms
- 26468 KB of logical-log spaces (or 6617 pages) for 4 K page platforms

During migration, a second database, the **sysadmin** database, is created in the **root** dbspace. As you work after migrating, the **sysadmin** database, could grow dramatically. You can move the **sysadmin** database to a different dbspace.

You might need to increase the physical log size to accommodate new features, and you might consider adding a new chunk.

If your migration fails because there is insufficient space in the partition header page, you must unload your data before you try to migrate again. Then you must manually load the data into the new version.

The root chunk should contain at least ten percent free space when converting to the new version of the server.

In some cases, even if the database server migration is successful, internal conversion of some databases might fail because of insufficient space for system catalog tables. For more information, see the release notes for this version of Informix®.

Add any additional free space to the system prior to the migration. If the dbspaces are nearly full, add space before you start the migration procedure. When you start the new version of Informix® on the same root dbspace of the earlier database server, Informix® automatically converts the **sysmaster** database and then each database individually.

For a successful conversion of each database, ensure that 2000 KB of free space per database is available in each dbspace where a database resides.

To ensure enough free space is available:

1. Calculate the amount of free space that each dbspace requires.

In the following equation, n is the number of databases in the dbspace and X is the amount of free space they require:

$$X \text{ kilobytes free space} = 2000 \text{ kilobytes} * n$$

The minimum number of databases is 2 (for the **sysmaster** and **sysadmin** databases).

2. Check the amount of free space in each dbspace to determine whether you need to add more space.

You can run SQL statements to determine the free space that each dbspace requires and the free space available. These statements return the free-space calculation in page-size units. The **free_space_req** column value is the free-space requirement, and the **free_space_avail** column value is the free space available.

The following SQL statement shows how to determine the free space that each dbspace requires:

```
DATABASE sysmaster;
SELECT partdbsnum(partnum) dbspace_num,
       trunc(count(*) * 2000) free_space_req
  FROM sysdatabases
 GROUP BY 1
 ORDER BY 1;
```

The following SQL statement queries the **syschunks** table and displays the free space available for each dbspace:

```
SELECT dbsnum dbspace_num, sum(nfree) free_space_avail
  FROM syschunks
 GROUP BY 1
 ORDER BY 1;
```



Important: If less free space is available than the dbspace requires, either move a table from the dbspace to another dbspace or add a chunk to the dbspace.

The dbspace estimates could be higher if you have an unusually large number of SPL routines or indexes in the database.

Related information

[Check and configure available space for reversion on page 53](#)

Configuring for recovery of restore point data in case an upgrade fails

By default, the `CONVERSION_GUARD` configuration parameter is enabled and a temporary directory is specified in the `RESTORE_POINT_DIR` configuration parameter. These configuration parameters specify information that Informix® can use if an upgrade fails. You can change the default values of these configuration parameters before beginning an upgrade.

Before you begin

Prerequisites: The directory specified in the `RESTORE_POINT_DIR` configuration parameter must be empty before the upgrade begins, but not when recovering from a failed update.



Important:

After a failed upgrade, do not empty the `RESTORE_POINT_DIR` directory before you attempt to run the `onrestorept` utility. The server must be offline after a failed upgrade.

About this task

You can change the value of the `CONVERSION_GUARD` configuration parameter or the directory for restore point files before beginning an upgrade. The default value for the `CONVERSION_GUARD` configuration parameter in the `ONCONFIG` file is `2`, and the default directory where the server will store the restore point data is `$INFORMIXDIR/tmp`. You must change this information before beginning an upgrade. You cannot change it during an upgrade.

To change information:

1. If necessary for your environment, change the value of the `CONVERSION_GUARD` configuration parameter.

When the `CONVERSION_GUARD` configuration parameter is set to `2` (the default value), the server will continue the upgrade even if an error related to capturing restore point data occurs, for example, because the server has insufficient space to store the restore point data.

However, if the `CONVERSION_GUARD` configuration parameter is set to `2` and the upgrade to the new version of the server fails, you can use the **onrestorept** utility to restore your data.

However, if you set the `CONVERSION_GUARD` configuration parameter to `2`, conversion guard operations fail (for example, because the server has insufficient space to store restore point data), and the upgrade to the new version fails, you cannot use the **onrestorept** utility to restore your data.

2. In the RESTORE_POINT_DIR configuration parameter, specify the complete path name for a directory that will store restore point files.

The server will store restore point files in a subdirectory of the specified directory, with the server number as the subdirectory name.

What to do next

If the CONVERSION_GUARD configuration parameter is set to `1` and an upgrade fails, you can run the **onrestorept** utility to restore the Informix® instance back to its original state just before the start of the upgrade.

If the CONVERSION_GUARD configuration parameter is set to `1` and conversion guard operations fail (for example, because the server has insufficient space to store restore point data), the upgrade to the new version will also fail.

If any restore point files from a previous upgrade exist, you must remove them before you begin an upgrade.

Even if you enable the CONVERSION_GUARD configuration parameter, you should still make level 0 backup of your files in case you need to revert after a successful upgrade or in case a catastrophic error occurs and you cannot revert.

Saving copies of the current configuration files

Save copies of the configuration files that exist for each instance of your source database server. Keep the copies available in case you decide to use the files after migrating or you need to revert to the source database server.

Although you can use an old ONCONFIG configuration file with Informix® Version 14.10, you should use the new Version 14.10 ONCONFIG file, or at least examine the file for new parameters.

Configuration files that you might have are listed in [Table 2: Configuration files to save from the source database server on page 11](#).

Table 2. Configuration files to save from the source database server

UNIX™ or Linux™	Windows™
<code>\$INFORMIXDIR/etc/\$ONCONFIG</code>	<code>%INFORMIXDIR%\etc\%ONCONFIG%</code>
<code>\$INFORMIXDIR/etc/onconfig.std</code>	<code>%INFORMIXDIR%\etc\onconfig.std</code>
<code>\$INFORMIXDIR/etc/oncfg*</code>	<code>%INFORMIXDIR%\etc\oncfg*</code>
<code>\$INFORMIXDIR/etc/sm_versions</code>	<code>%INFORMIXDIR%\etc\sm_versions</code>
<code>\$INFORMIXDIR/aaodir/adtcfg</code>	<code>%INFORMIXDIR%\aaodir\adtcfg.*</code>
<code>\$INFORMIXDIR/dbssodir/adtmasks</code>	<code>%INFORMIXDIR%\dbssodir\adtmasks.*</code>
<code>\$INFORMIXDIR/etc/sqlhosts</code> or <code>\$INFORMIXSQLHOSTS</code>	<code>%INFORMIXDIR%\etc\sqlhosts</code> or <code>\$INFORMIXSQLHOSTS</code>
<code>\$INFORMIXDIR/etc/tctermcap</code>	
<code>\$INFORMIXDIR/etc/termcap</code>	

If you use ON-Bar to back up your source database server and the logical logs, you must also save a copy of any important storage manager files and the following file:

UNIX™ or Linux™:

`$INFORMIXDIR/etc/ixbar.servernum`

Windows™:

`%INFORMIXDIR%\etc\ixbar.servernum`

The Informix® Primary Storage Manager does not use the `sm_versions` file. If you plan to use the Informix® Primary Storage Manager, you do not need the `sm_versions` file. However, if you use the Spectrum Protect or a third-party storage manager, you do need the `sm_versions` file.

If you are using a different directory as **INFORMIXDIR** for the new database server, copy `sm_versions` to the new **`$INFORMIXDIR/etc`**, or copy `sm_versions.std` to `sm_versions` in the new directory, and then edit the `sm_versions` file with appropriate values before starting the migration.

Preparing 12.10 BSON columns with DATE fields for upgrade

Before you upgrade from Informix® 12.10 you must unload binary JSON (BSON) columns with DATE fields into JSON format so that you can load them into Informix® 14.10

About this task

Perform the following steps on the 12.10 server.

1. Create an external table with a similar name as the original table and with JSON (instead of BSON) format for the date.

For example, assume that the original table named **datetab** has a BSON column named `i` that has DATE fields in it.

Use the following statement to create an empty, external table named **ext_datetab** that has a JSON column with DATE fields. The `DATAFILES` clause specifies the location and name of the delimited data file, which in this example is `disk:/tmp/dat.unl`.

```
create external table ext_datetab (j int, i json) using
  (datafiles ("disk:/tmp/dat.unl"),
   format "delimited");
```

2. Unload the data from the original table into the external table.

For example:

```
insert into ext_datetab select j, i::json from datetab;
```

What to do next

Complete other pre-migration steps. After you upgrade to the new server, you must load the JSON columns with DATE fields from the external table into a new table in BSON format.

Related information

[Finish preparing earlier versions of 12.10 databases for JSON compatibility on page 47](#)

Closing all transactions and shutting down the source database server

Before migrating, terminate all database server processes and shut down your source database server. This lets users exit and shuts down the database server gracefully. If you have long running sessions, you must also shut those down.

Inform client users that migration time is typically five to ten minutes. However, if migration fails, you must restore from a level-0 backup, so ensure that you include this possibility when you estimate how long the server will be offline.

Before you migrate from the original source database server, make sure that no open transactions exist. Otherwise, fast recovery will fail when rolling back open transactions during the migration.

To let users exit and shut down the database server gracefully

1. Run the **onmode -sy** command to put the database server in quiescent mode.
2. Wait for all users to exit.
3. Run the **onmode -l** command to move to the next logical log.
4. Run the **onmode -c** to force a checkpoint.
5. Make a level-0 backup of the database server.
6. Run the **ontape -a** command after the level-0 backup is complete.
7. Run the **onmode -yuk** command to shut down the system.

If you need to perform an immediate shutdown of the database server, run these commands:

```
onmode -l
onmode -c
onmode -ky
```

Initiating fast recovery to verify that no open transactions exist

A shutdown procedure does not guarantee a rollback of all open transactions. To guarantee that the source database server has no open transactions, put the source database server in quiescent mode and initiate fast recovery.

Run the following command to enter quiescent mode and initiate a fast recovery:

```
oninit -s
```

UNIX/Linux Only

On UNIX™ or Linux™, the **oninit -s** command rolls forward all committed transactions and rolls back all incomplete transactions since the last checkpoint and then leaves a new checkpoint record in the log with no open transactions pending.

You must run the **oninit -s** command before you initialize the new version of Informix®. If any transactions remain when you try to initialize the new database server, the following error message appears when you try to initialize the new database server, and the server goes offline:

```
An open transaction was detected when the database server changed log versions.
Start the previous version of the database server in quiescent mode and then shut
down the server gracefully, before migrating to this version of the server.
```

For more information about fast recovery, see your *Informix® Administrator's Guide*.

After you put the database server in quiescent mode and initiate fast recovery, issue the **onmode -yuk** command to shut down the database server. Then review the **online.log** file for any possible problems and fix them.

Only after proper shutdown can you bring the new database server (Informix® Version 14.10) through the migration path. Any transaction that is open during the migration causes an execution failure in fast recovery.

Verifying the integrity of the data

After verifying that no open transactions exist, verify the integrity of your data by running the **oncheck** utility. You can also verify the integrity of the reserve pages, extents, system catalog tables, data, and indexes. If you find any problems with the data, fix the problems before you make a final backup of the source database server.

To obtain the database names, use the following statements with DB-Access:

```
DATABASE sysmaster;
SELECT name FROM sysdatabases;
```

Alternatively, to obtain the database names, run the **oncheck -cc** command without any arguments and filter the result to remove unwanted lines, as shown in this example:

```
oncheck -cc | grep "ting database"
```

[Table 3: Commands for verifying the data integrity on page 14](#) lists the **oncheck** commands that verify the data integrity.

Table 3. Commands for verifying the data integrity

Action	oncheck Command
Check reserve pages	oncheck -cr
Check extents	oncheck -ce
Check system catalog tables	oncheck -cc <i>database_name</i>
Check data	oncheck -cD <i>database_name</i>
Check indexes	oncheck -cl <i>database_name</i>

Verifying that the database server is in quiescent mode

Before you make a final backup, verify that your source database server is in quiescent mode.

Run the `onstat -` command to verify that the database server is in quiescent mode.

The first line of the `onstat` output shows the status of your source database server. If the server is in quiescent mode, the status line includes this information:

```
Quiescent -- Up
```

Making a final backup of the source database server

Use ON-Bar or **ontape** to make a level-0 backup of the source database server, including all storage spaces and all used logs. After you make a level-0 backup, also perform a complete backup of the logical log, including the current logical-log file.

Be sure to retain and properly label the tape volume that contains the backup.



Important: You must also make a final backup of each source database server instance that you plan to convert.

For ON-Bar, remove the **ixbar** file, if any, from the `$INFORMIXDIR%/etc` or `%INFORMIXDIR%\etc` directory after the final backup. Removing the **ixbar** file ensures that backups for the original source database server are not confused with backups about to be done for the new database server. Follow the instructions regarding expiration in your storage manager documentation.

For more information about making backups, see the *Informix® Backup and Restore Guide*.

Verifying that the source database server is offline

Before you install the new database server, verify that the source database server is offline. You must do this because the new database server uses the same files.

You cannot install the new database server if any of the files that it uses are active.

You can also use the `onstat` utility to determine that shared memory was not initialized.

Pre-migration checklist of diagnostic information

Before you migrate to a newer version of Informix®, gather diagnostic information, especially if you have large, complex applications. This information will be useful to verify database server behavior after migration. This information will also be useful if you need help from HCL Software Support.

If you have problems, you or HCL Software Support can compare the information that you gather with information obtained after migration.

The following table contains a list of the diagnostic information that you can gather. You can print the checklist. Then, after you get the information specified in each row, check the second column of the row.

Table 4. Checklist of information to get before migrating

Information to Get Before Migrating	Done
Get the SQL query plans for all regularly used queries, especially complex queries, by using SET EXPLAIN ON.	
Run the dbschema -d -hd command for all critical tables. The output contains distribution information.	
Get oncheck -pr output that dumps all of the root reserved pages.	
Make a copy of the ONCONFIG configuration file. A copy of the ONCONFIG file is essential if you need to revert to an earlier version of the database server. In addition, a copy of this file is useful because oncheck -pr does not dump all of the configuration parameters.	
Prepare a list of all the environment variables that are set using the env command.	
During times of peak usage: <ul style="list-style-type: none"> • Obtain an online.log snippet, with some checkpoint durations in it • Run onstat -aF, -g all, and -g stk all. 	
During times of peak usage, run the following onstat commands repeatedly with the -r repeat option for a period of about three to five minutes: <ul style="list-style-type: none"> • onstat -u, to see the total number of sqlxecs used • onstat -p, for read and write cache rates, to detect deadlocks and the number of sequential scans • onstat -g nta, a consolidated output of -g ntu, ntt, ntm and ntd • onstat -g nsc, -g nsd, and -g nss for the status of shared memory connections • onstat -P, -g tpf, and -g ppf • vmstat, iostat and sar, for cpu utilization • timex of all queries that you regularly run 	

Related information

[Preparing for migration on page 7](#)

Migrating from 32-bit to 64-bit database servers

If you are migrating from a 32-bit version of Informix® to a 64-bit version of Informix® or reverting from a 64-bit version of Informix®, you might need to follow additional steps to update certain internal tables.

These steps are documented in the platform-specific machine notes that are provided with your database server.

For 32- to 64-bit migrations, change SHMBASE and STACKSIZE according to the `onconfig.std` configuration file for the new version.

All UDRs and DataBlade® modules that were built in 32-bit mode must be recompiled in 64-bit mode because they will not work with the 64-bit database server. If you have any UDRs that were developed in 32-bit mode, make sure that proper size and alignment of the data structures are used to work correctly on a 64-bit computer after recompiling in 64-bit mode. For more information, refer to the machine notes.

Migrating from 32-bit to 64-bit with collection types that use the SMALLINT data type

If you are moving your database from a 32-bit computer to a 64-bit computer and your database contains collection types that use the SMALLINT data type, you must take extra steps to prevent memory corruption. Collection types are the ROW, LIST, SET, and MULTISSET data types. This restriction applies if you are upgrading from an older version of Informix® on 32-bit to the current version of Informix® on 64-bit, or if you are moving from 32-bit to 64-bit on the current version of Informix®.

To migrate a database with SMALLINT collection types from 32-bit to 64-bit, use one of the following methods:

- Export and import the data.
 1. Export the data from the 32-bit computer.
 2. Import the data onto the 64-bit computer.
- Drop and recreate specific collection types and database objects.
 1. Drop the collection types that use the SMALLINT data type and all other database objects that reference them (such as tables, columns, SPL routines, triggers, indexes, and so on).
 2. Recreate the collections types and all the other necessary database objects.

Enterprise Replication and migration

You must coordinate the migration of all servers that are involved in data replication.

These topics describe the additional tasks that you must perform when migrating to and reverting from Informix® Version 14.10 if you are running Enterprise Replication.

Related information

Upgrading Informix (in-place migration)

Types of migration

Migrating Informix (non-in-place migration)

Preparing to migrate with Enterprise Replication

If you use Enterprise Replication, you must do replication-related tasks to prepare for migration.

Before you begin

You must do all migration operations as user **informix**, unless otherwise noted.

Only a DBSA can run the `cdr check queue` command. With a non-root installation, the user who installs the server is the equivalent of the DBSA, unless the user delegates DBSA privileges to a different user.

To prepare for migration with Enterprise Replication:

1. Stop applications that are performing replicable transactions.
2. Make sure that the replication queues are empty.

If you are migrating from Informix® 12.10.xC1 or later releases, run the following commands to check for queued messages and transactions:

- a. `cdr check queue -q cntrlq targetserver`
- b. `cdr check queue -q sendq targetserver`
- c. `cdr check queue -q recvq targetserver`

If you are migrating from an earlier version of Informix®, run the following commands:

- a. Run `onstat -g grp` to ensure that the Enterprise Replication grouper does not have any pending transactions. The grouper evaluates the log records, rebuilds the individual log records in to the original transaction, packages the transaction, and queues the transaction for transmission.
 - b. Run `onstat -g rqm` to check for queued messages.
3. Shut down Enterprise Replication by running the `cdr stop` command.

Results

Now you can complete the steps in [Preparing for migration on page 7](#) and, if necessary, in [Migrating from 32-bit to 64-bit database servers on page 16](#).

Migrating with Enterprise Replication

If you use Enterprise Replication, you must complete replication-related tasks when you migrate to a new version of Informix®. If you are converting to Informix® 12.10.xC4 or later from an earlier fix pack in the same release, you also have to complete these tasks.

About this task

Prerequisites:

- Complete the steps in [Preparing for migration on page 7](#).
- Perform all migration operations as user **informix**.
- All servers in the Enterprise Replication domain must be available.

To migrate with Enterprise Replication:



Note:



If you are migrating server from 12.10xC4 or a later version to 14.10xC6 or a later version, after upgrading server to 14.10FC6 or a later version, for the first time when you start ER, make sure to start ER using `cdr cleanstart` command to force required schema changes to `syscdr` database. Rest of the steps can be skipped.

1. Perform the tasks that are described in [Migrating to the new version of Informix on page 40](#), including starting the new version of the server.
2. For each node involved in Enterprise Replication, back up the **syscdr** databases by using the `dbexport -ss` command or the `dbschema -ss` command and the `UNLOAD` statement, or by a combination of these methods.
The `-ss` option prevents backup tables from using default extent sizes and row-level locking, which is not an appropriate lock mode with Enterprise Replication.
3. Make sure that no replicable transactions occur before Enterprise Replication starts.
4. If you are upgrading to a new release, run the conversion script, named `concdr.sh`, in the `$INFORMIXDIR/etc/conv` directory on UNIX™, or `concdr.bat`, in the `%INFORMIXDIR%\etc\conv` directory on Windows™. Do not run the script when you migrate between fix packs of the same release except where noted.

To convert to 12.10.xC4 or later:

```
UNIX™: % sh concdr.sh from_version 12.10.xC4
```

```
Windows™: concdr.bat from_version 12.10.xC4
```

The valid `from_version` values are: `12.10.xC3`, `12.10.xC2`, `12.10.xC1`, `11.70`, `11.50`, `11.10`, and `10.00`.

Because version 12.10.xC4 is the latest fix pack with conversion, specify `12.10.xC4` even if you are upgrading to later a fix pack version, such as 12.10.xC6.

To convert to earlier fix packs of 12.10:

```
UNIX™: % sh concdr.sh from_version 12.10
```

```
Windows™: concdr.bat from_version 12.10
```

The valid `from_version` values are: `11.70`, `11.50`, `11.10`, and `10.00`.

5. Ensure conversion completed successfully.

The script prints messages and conversion details to standard output, and stores the information in the following file:

- UNIX™: `$INFORMIXDIR/etc/concdr.out`
- Windows™: `%INFORMIXDIR%\etc\concdr.out`



Important: If you receive the following message, you must resolve the problems reported in the `concdr.out` file, restore the **syscdr** database from backup, and then start from step 1.

```
'syscdr' conversion failed.
```

When you receive the following message, conversion is complete and you can go to the next step.

```
'syscdr' conversion completed successfully.
```

6. After successful conversion, if you are upgrading server to 14.10xC6 or later version, start Enterprise Replication by running the `cdr cleanstart` command, otherwise run `cdr start` command.

7. If you upgraded the servers in a grid from version 11.70 to version 12.10 and you want to copy external files to the servers in the grid, you must enable the ability to copy external files. To enable the copying of external files, run this command:

```
cdr modify grid grid_name --enablegridcopy server_groupname
```

Results



Important: After you convert to the new version of Informix® with Enterprise Replication, do not drop the **syscdr** database. If **syscdr** is dropped, you cannot revert to the older database server with Enterprise Replication because the data required to carry out the reversion is stored in the **syscdr** database.

Reverting with Enterprise Replication

If you use Enterprise Replication, you must complete replication-related tasks when you revert from the new version of Informix®. If you are reverting from Informix® 12.10.xC4 or later to a fix pack earlier than 12.10.xC4 in the same release, you also have to complete these tasks.

Before you begin

Prerequisites:

- Perform all reversion operations as user **informix**.
- Enterprise Replication must be running, or you must delete the replication server before you revert.
- All servers in the Enterprise Replication domain must be available.
- Replication queues must be nearly empty.

About this task

If you want to revert to a version earlier than when Enterprise Replication was defined on this server, you must remove Enterprise Replication from this server before reverting. After you remove Enterprise Replication, you can revert your server using the instructions at [Reverting from Informix Version 14.10 on page 51](#).

To revert from Version 14.10 with Enterprise Replication:

1. Release or remove any grid statements that are deferred from propagation by running the `ifx_grid_release()` or `ifx_grid_remove()` function.
2. Stop applications that are doing replicable transactions.
3. Remove Enterprise Replication features from the current release that cannot be reverted. For more information, see [Reversion requirements and limitations on page 52](#).
4. Run the `onstat -g cat repls` command to check whether if Enterprise Replication is in alter mode. If so, run the `cdr alter -off` command.
5. Delete shadow replicates.
6. Back up the **syscdr** databases with **dbschema** or UNLOAD.

7. Run the reversion script, named `revcdr.sh`, in the `$INFORMIXDIR/etc/conv` directory on UNIX™, or `revcdr.bat`, in the `%INFORMIXDIR%\etc\conv` directory on Windows™:

To revert from 12.10.xC4 or later to an earlier version:

```
UNIX™: % sh revcdr.sh 12.10.xC4 to_version
```

```
Windows™: revcdr.bat 12.10.xC4 to_version
```

Because version 12.10.xC4 is the latest fix pack with conversion, specify `12.10.xC4` even if you are reverting from a later fix pack version, such as 12.10.xC6.

Valid `to_version` values are `12.10.xC3`, `12.10.xC2`, `12.10.xC1`, `11.70`, `11.50`, `11.10`, and `10.00`. You do not have to specify a fix pack level except where noted.

To revert from earlier 12.10 fix packs:

```
UNIX™: % sh revcdr.sh 12.10 to_version
```

```
Windows™: revcdr.bat 12.10 to_version
```

Valid `to_version` values are `11.70`, `11.50`, `11.10`, and `10.00`.

This script runs a reversion test followed by the actual Enterprise Replication reversion.

Result



Important: If the reversion test or actual reversion fails, check the file `$INFORMIXDIR/etc/revtestcdr.out` or `revcdr.out`. Resolve any problems before going to the next step.

8. Perform database server reversion tasks, as described in [Reverting from Informix Version 14.10 on page 57](#).
9. Run `onmode -l` and `onmode -c` to prevent the database server from failing when you start Enterprise Replication.
10. Start Enterprise Replication by running the `cdr start` command.

Related information

[Preparing to revert on page 51](#)

High-availability cluster migration

You must coordinate the migration and reversion of all servers that are involved in high-availability clusters. You can use a rolling upgrade in some cases to update servers while the cluster is online, with minimal interruption to client applications. Otherwise, you must schedule cluster downtime to upgrade the servers. A cluster must be offline to reverse an upgrade or revert to an earlier release.

A rolling upgrade can minimize downtime, but the tradeoff is that you must spend time to prepare for a rolling upgrade. The effort to prepare your servers for a rolling upgrade depends on how closely your current configuration matches the procedure requirements. In some cases, you might find it easier to plan for downtime during a period of low activity instead of setting up your environment for a rolling upgrade. The downtime that is required depends on your system, but smaller clusters usually require less downtime.

Use the following table to help you determine which upgrade procedure to use. Also, review the requirements and limitations that are documented in each procedure.

Table 5. Comparison of upgrade procedures for high-availability clusters

Upgrade to	Procedure	Use	Procedure overview
Next consecutive fix pack or PID of the same major version	Rolling upgrade of online cluster to the next fix pack or PID (UNIX, Linux) on	As of Informix® 12.10.xC5, you can apply the next consecutive fix pack or interim fix (PID) of the same major version to all the servers while the cluster remains online.	You must stop and restart each server in the cluster, including the primary. Specifically, you upgrade each secondary server, and then you stop the primary server and promote an upgraded secondary server to primary. Next, you upgrade the original primary server, bring it online as a secondary server, and promote it back to primary, if necessary. Interruption to client applications is minimized because transactions are redirected to active servers by Connection Manager or through the client applications. The new database server capabilities can be used in the cluster after all servers are upgraded.
Any later fix pack or PID of the same major version	Upgrading an offline cluster to a fix pack or PID on	<p>You can use this procedure within a major version to apply a fix pack or PID in offline clusters in the following situations:</p> <ul style="list-style-type: none"> • Your cluster does not meet the requirements for a rolling upgrade. • The limitations that are imposed by a rolling upgrade are impractical for your environment. • You prefer or need to bring the servers in the cluster offline before you upgrade them. 	You stop all servers in the cluster during a period when downtime is acceptable. While the cluster is offline, you upgrade all the servers. You then start the primary followed by the secondary servers. You don't have to rebuild the cluster or clone the servers.

Table 5. Comparison of upgrade procedures for high-availability clusters

(continued)

Upgrade to	Procedure	Use	Procedure overview
New major version	Migration offline cluster to a new major version	You can use this procedure to migrate the servers in your offline cluster to a new major version. For example, you can migrate from Informix® 12.10 to Informix® 14.10.	You stop the servers in a specific order during a period when planned downtime is acceptable. Then, if you are migrating from Informix 12.10 or later version, You stop all servers in the cluster during a period when downtime is acceptable. While the cluster is offline, you upgrade all the servers. You then start the primary followed by the secondary servers. You don't have to rebuild the cluster or clone the servers.
A fix pack or PID that requires conversion	Applying a new major version package 29	Also, you can use this procedure to apply a fix pack or PID that requires standard conversion procedures. Usually conversion is not required for a fix pack or PID. The machine notes indicate whether conversion is required.	If you are migrating from Informix® 11.50 or older version, you migrate only the primary server. During migration, the database server automatically removes secondary servers. After you migrate the primary server, you must rebuild the cluster by recreating all secondary servers. You can use the ifxclone utility to recreate secondary servers.
Any supported release	Rolling upgrade of an online cluster with Enterprise Replication package 31	This alternative rolling upgrade procedure requires a working knowledge of Enterprise Replication. You can use this procedure to upgrade online clusters to any supported product release. You can use this procedure to upgrade to a new major version or to apply any fix pack or PID (not just the next consecutive one) within a release. This procedure is more flexible than the other rolling upgrade procedure; however, it is more complex to set up.	You temporarily convert the primary and secondary servers to stand-alone Enterprise Replication servers. The upgrade occurs without incurring any downtime because Enterprise Replication supports replication between different versions of the server software.

Related information

Upgrading Informix (in-place migration)

Types of migration

Migrating Informix (non-in-place migration)

Preparing to migrate, upgrade, or revert clusters

If you use high-availability clusters, you must coordinate the migration of all of the servers that are involved in a high-availability cluster, and you must perform additional steps when preparing to migrate.

About this task

Prerequisites:

- Perform all migration operations as user **informix**.
- Download the product package from [at](#) , or download [recommended fixes for server products](#).
- If you are migrating to a new version of the server, complete all steps in [Preparing for migration on page 7](#).

To prepare for migration or reversion of a cluster:

1. Prepare each server in the cluster:

- a. Install the target server software in a different location from where the source database server software is installed. Do not install the target server software over the source server.



Tip: Install the new version of Informix® Client Software Development Kit (Client SDK), which contains the Connection Manager that you must use with the new version of the server. If you want to use only the Connection Manager from the new version of Client SDK, install Client SDK in a new location and use the Connection Manager from that location.

- b. Copy the `onconfig` and `sqlhosts` configuration files to the target installation directory (for example, **\$INFORMIXDIR/etc**).


- c. Install on the target server any user-defined objects or DataBlade® modules that are used on the source server.

2. Back up your primary server. You can perform this step in the following ways:

Choose from:

- Use ON-Bar or ontape to make a level-0 backup on the primary source server.
- If you have a high-availability cluster with a High-availability Data Replication (HDR) secondary server, you can use the HDR secondary server as a standby server for any contingencies that occur while you upgrade the primary server. However, if it is necessary to use an HDR secondary server as a standby server for contingencies, do not perform updates on the standby server while migration or reversion is in progress,

because the updates cannot be replicated and will be lost. Additionally, nonlogged objects on the primary server will not exist on the secondary server.

 **Attention:** Do not use RS secondary servers as backup servers, because transactions could be lost.

Rolling upgrade of an online cluster to the next fix pack or PID (UNIX, Linux)

As of Informix® 12.10.xC5, you can apply the next consecutive fix pack or interim fix (PID) to your current version of the database server. During the rolling upgrade, the cluster remains online even though the servers in the cluster run different levels of the software. The new capabilities can be used in the cluster after all the servers in the cluster are upgraded.

About this procedure

This rolling upgrade procedure works only on UNIX and Linux platforms. Use this procedure to apply the next *consecutive* fix pack or PID in the current major version. For example, you can do a rolling upgrade from 12.10.xC4 to 12.10.xC5. Otherwise, you must use a different procedure that is documented at [High-availability cluster migration on page 21](#).

Do *not* use this procedure in the following situations:

- To apply a fix pack or PID that requires conversion.
- To upgrade to 12.10.xC5 from 12.10.xC3, 12.10.xC2, or 12.10.xC1.
- To upgrade an earlier major version of the server to a later major version of the database server, for example, to upgrade from version 11.70 to 12.10.
- To upgrade from a patch release or special build, unless advised to do so by Software Support.

You must stop and restart each server in the cluster, including the primary, as part of this procedure. Specifically, you upgrade each secondary server, and then you stop the primary server and promote an upgraded secondary server to primary. Next, you upgrade the original primary server, bring it online as a secondary server, and promote it back to primary, if necessary.

You must have the following permission on all the servers in the cluster: user **root** or user **informix**.

This procedure consists of these steps:

- [Prepare for a rolling upgrade on page 25](#)
- [Upgrade the servers on page 26](#)
- [Return the cluster to its original configuration on page 27](#)

Prepare for a rolling upgrade

Use the following steps to plan for and prepare the servers for rolling upgrade:

1. Complete the steps in [Preparing to migrate, upgrade, or revert clusters on page 24](#), which include installing the new software on all the servers in the cluster, copying the appropriate configuration files, and backing up the primary server.
2. If you are migrating from a server version prior to 14.10.xC8, or from a Client SDK version prior to 4.50.xC8, you must upgrade all Connection Managers before upgrading any server in your cluster. Upgrade them one at a time to avoid a service interruption that would affect your applications. All Connection Managers should be running with version 4.50.xC8 or later before continuing to the next step.
3. Configure client redirection to minimize interruption of service.

Set up redirection and connectivity for clients by using the method that works best for your environment.

If Connection Manager controls the connection redirection in the cluster: Ensure that every service level agreement (SLA) definition in the Connection Manager configuration file can redirect to at least one server other than the one you are about to update. For example, assume that you have an SLA with only one secondary. Before you upgrade the secondary server in that SLA, update the SLA to include the cluster primary (PRI).

4. Ensure that the primary server has an appropriate amount of disk space for the logical log records that are created during the entire upgrade process. The space that is required depends on your environment.



Attention: If a log wrap occurs during the rolling upgrade procedure, you must apply the fix pack or PID while the cluster is offline.



Tip: Examine the online log to get an estimate of your data activity during normal operations. You might want to ensure that you have enough space for data activity for a day. Also, you might find it convenient to plan the rolling upgrade for a period of low traffic.

5. Prepare the secondary server that will become the primary when you upgrade the original primary server. You must use an SD secondary or a fully synchronous HDR secondary server that has transactional consistency with the original primary server.
 - a. If the cluster contains an SD secondary server, you don't need to do any additional preparation to that server.
 - b. If the cluster contains an HDR secondary server, make sure that it runs in fully synchronous (SYNC) mode.
 - c. If the cluster contains only RS secondary servers in addition to the primary server, you must change one of the RS secondary servers to an HDR secondary server in SYNC mode.

Upgrade the servers

Follow the steps in this section to upgrade the cluster servers in the following order:

1. Remote standalone (RS) secondary server
2. HDR secondary server
3. Shared disk (SD) secondary server
4. Primary server

! **Important:** Upgrade the primary server only after *all* the secondary servers are upgraded and tested. After you upgrade the primary server, if you want to revert to your original environment you must take the cluster offline.

1. Run the `onmode -c` command to force a checkpoint for each server.
2. If a wire listener is running on the server that you want to upgrade, stop that wire listener.
3. Stop the server that you want to upgrade. If you can wait for all connections to exit gracefully before stopping the server, use the `onmode -kuy` command. Otherwise, use the `onmode -ky` command to stop the server.

- **When you stop a secondary server:** If [redirection is configured on page 26](#) for the cluster, the client application automatically connects to another active server in the cluster.
- **When you stop the primary server:** If failover is configured for the cluster, a secondary server is promoted automatically to primary. Otherwise, you can run the `onmode -d make primary` command to promote a prepared secondary server to primary.

i **Tip:** If the primary is offline before the failover, you must use the `onmode -d make primary force` command.

4. Set your environment to use the fix pack or PID that you installed on the server.
 - a. Set the **INFORMIXDIR** environment variable to the full path name for the target installation.
 - b. Update all environment variables that depend on the **INFORMIXDIR** environment variable. At a minimum, update these environment variables: **PATH**, **DBLANG**, **INFORMIXSQLHOSTS**, and any platform-specific library path environment variables, such as **LD_LIBRARY_PATH**.
5. Start the upgraded server.

To start an upgraded secondary server: Use the `oninit` command.

To start an upgraded original primary server: Start the original primary server as a secondary server. For convenience, start it as the server type that was promoted to primary during the rolling upgrade. For example, if you promoted an HDR server to primary for the rolling upgrade, start the original primary as an HDR secondary server.

- To start the upgraded server as an SD secondary server, run the `oninit -SDS` command.
- To start the upgraded server as an HDR secondary server, run the `oninit -PHY` command, and then run the following command: `onmode -d secondary primary_server secondary_server`

After the server starts, it runs the new version of the software and automatically reconnects to the cluster.

6. To verify that the upgraded secondary server is active in the cluster, run the `onstat -g cluster` command on both the server you upgraded and on the primary server.
7. If you stopped the wire listener to upgrade this server, restart the wire listener.

After you upgrade all the servers and restart them, the original primary server is running as a secondary server.

Return the cluster to its original configuration

Use the following steps if you want the cluster to operate as it did before you prepared it for a rolling upgrade.

1. Manually promote the secondary server that was the original primary to be the primary server again.
 - a. Run the `onmode -c` command to force a checkpoint.
 - b. Run the `onmode -d make primary` command to promote the secondary server to primary.
2. Undo any changes that you made when you prepared the servers for a rolling upgrade. Some of these optional steps might not apply to you.
 - Adjust the amount of disk space that is allocated for logical log records.
 - Convert the HDR secondary server back to an RS secondary server.
 - Change the HDR secondary server back to ASYNC mode from SYNC mode.
 - Change the Connection Manager SLA definitions.

Related information

[Upgrading an offline cluster to a fix pack or PID on page 28](#)

Upgrading an offline cluster to a fix pack or PID

You can bring a high-availability cluster offline to apply any PID or fix pack of the same version of the database server. For example, you must bring the clusters offline to upgrade the servers to the current fix pack if earlier fix packs were not applied to the server. This procedure is an alternative to the rolling upgrade procedure.

Before you begin

Prerequisites:

- Verify that you are upgrading to a PID or fix pack in which standard conversion procedures are not necessary. If the PID or fix pack requires you to complete standard conversion procedures, or if you want to upgrade to a new major release, go to [Migrating an offline cluster to a new major version on page 29](#) instead of following the procedures in this topic.
- Complete the steps in [Preparing to migrate, upgrade, or revert clusters on page 24](#).
- Perform all migration operations as user **informix**.

To upgrade offline clusters to a new PID or fix pack:

1. Stop the Connection Manager by issuing the `oncmsh -k connection_manager_name` command.
2. If you are using a High-availability Data Replication (HDR) secondary server as a backup server in case of contingencies:
 - a. Quiesce the primary server by issuing an `onmode -sy` command to prevent user connections to the server.
 - b. Force a checkpoint by issuing an `onmode -c` command on the primary server.
3. Stop secondary servers in the cluster in the following order:

- a. If you have remote standalone (RS) servers, stop them by issuing the onmode -ky command.
- b. If you have shared disk (SD) servers, stop them by issuing the onmode -ky command.
- c. If you have an HDR secondary server, stop it issuing the onmode -ky command.
4. Stop the primary server by issuing the onmode -ky command.
5. On each server, set the INFORMIXDIR environment variable to the full path name for the target installation.
6. Ensure that all of the necessary configuration files are available in the target installation.
7. Start the servers in the cluster and perform additional tasks in the following order:
 - a. Start the primary server by running an oninit command.
 - b. Wait for primary server to be in online (multi-user) mode.
 - c. Start the Connection Manager by running an oncmsm command.
 - d. Start the HDR secondary server by running an oninit command.
 - e. Start SD servers by running an oninit command.
 - f. Start RS servers by running an oninit command.

Related information

[Rolling upgrade of an online cluster to the next fix pack or PID \(UNIX, Linux\) on page 25](#)

Migrating an offline cluster to a new major version

If you are migrating from Informix 11.70 or later version to 14.10xC4 or later version, you stop all servers in the cluster during a period when downtime is acceptable. While the cluster is offline, you upgrade all the servers. You then start the primary followed by the secondary servers. You need not rebuild the cluster or clone the servers.

Before you begin

If you are migrating from Informix 11.50 or older version or migrating to 14.10xC3 or older version, you migrate only the primary server. During migration, the database server automatically removes secondary servers. After you migrate the primary server, you must rebuild the cluster by recreating all secondary servers. You can use the **ifxclone** utility to recreate secondary servers. This procedure also applies if you want to upgrade a cluster to a new PID or fix pack that requires standard conversion procedures.

Prerequisites:

- Complete the steps in [Preparing for migration on page 7](#).
- Complete the steps in [Preparing to migrate, upgrade, or revert clusters on page 24](#).
- Perform all migration operations as user **informix**.

About this task

Be sure to stop and start the servers in the cluster in the order shown in the following procedure.

To migrate to a new version with high-availability clusters:

1. Stop the Connection Manager by issuing the `oncmsm -k connection_manager_name` command.
2. If you are using a High-availability Data Replication (HDR) secondary server as a backup server in case of contingencies:
 - a. Quiesce the primary server by issuing an `onmode -sy` command to prevent user connections to the server.
 - b. Force a checkpoint by issuing an `onmode -c` command on the primary server.
3. Stop the secondary servers in the cluster in the following order:
 - a. If you have remote standalone (RS) secondary servers, stop them by issuing the `onmode -ky` command.
 - b. If you have shared disk (SD) servers, stop them by issuing the `onmode -ky` command.
 - c. If you have an HDR secondary server, stop it by issuing the `onmode -ky` command.
4. Stop the primary server by issuing the `onmode -ky` command.
5. On each server, set the `INFORMIXDIR` environment variable to the full path name for the target installation.
6. Ensure that all of the necessary configuration files are available in the target installation.
7. Optional: Enable quick reversion to a consistent restore point if the migration fails. Do this by setting the `CONVERSION_GUARD` and `RESTORE_POINT_DIR` configuration parameters. (For more information, see [Configuring for recovery of restore point data in case an upgrade fails on page 10.](#))
8. Start the primary server by issuing an `oninit` command.
9. Ensure that the conversion to the target server was successful and that the server is in multi-user mode.
10. If you are migrating from Informix 11.70 or later version to 14.10xC4 or later version, for HDR and RSS servers,
 - a. Enable quick reversion to a consistent restore point if the migration fails. Do this by setting the `CONVERSION_GUARD` and `RESTORE_POINT_DIR` configuration parameters. (For more information, see [Configuring for recovery of restore point data in case an upgrade fails on page 10.](#))
 - b. Start secondary server by issuing an `oninit` command.
 - c. Check `'onstat -g cluster'` output at primary server, and once HDR/RSS secondary server connected to primary server, force a checkpoint at primary server using `'onmode -c'` command and make sure checkpoint is replayed at HDR/RSS secondary server.
11. Start the Connection Manager by issuing an `oncmsm` command.
12. If you are migrating from pre-11.10 versions of Informix® and need SD secondary servers on the primary server in a shared-disk cluster, set the primary server by issuing the `onmode -d set SDS primary primary_server_name` command.
13. Start SD secondary servers by issuing `oninit` commands.
14. Start the servers in the cluster and perform additional tasks in the following order:
15. If you are migrating from Informix 11.50 or older version or migrating to 14.10xC3 or older version, for HDR ad RSS servers,
 - a. Back up all logs. Then use ON-Bar or **ontape** to make a level-0 backup on the primary server to use to reestablish the RS and HDR secondary servers if necessary.
 - b. If you have RS secondary servers:

- i. Add RS entries on the primary server by issuing **onmode -d add RSS** *rss_server_name* commands.
 - ii. Start RS secondary servers with level-0 restore operations from the level 0 backup that was made on the primary server after migration.
 - iii. On RS secondary servers, run the **onmode -d RSS** *primary_server_name* command, and wait for the "RSS secondary server operational" message to appear after each command.
- c. If you have an HDR secondary server:
- i. Reestablish the pair on the primary server by issuing an **onmode -d primary** *hdr_secondary_server_name* command.
 - ii. Start the HDR secondary server with level-0 restore operations from the level 0 backup that was made on the primary server after migration.
 - iii. On the HDR secondary server, run the **onmode -d secondary** *primary_server_name* command, and wait for the "**HDR secondary server operational**" message to appear after each command.
16. Perform any additional standard migration tasks described in [Migrating to the new version of Informix on page 40](#) and in [Completing required post-migration tasks on page 46](#).
17. If you have Connection Manager configuration files that you created with a version of Informix® Client Software Development Kit (Client SDK) that is prior to version 3.70.xC3, you must convert the files. You must convert them because the older files are incompatible with the current version of the Connection Manager. For more information, see [Converting older formats of the Connection Manager configuration file to the current format on page](#) .

Results

The migration of all servers in the cluster is now complete.



Note: Reversion is not supported for HDR and RSS secondary servers. You need to revert primary server, and you must rebuild the cluster by recreating all secondary servers. You can use the **ifxclone** utility to recreate secondary servers.

Rolling upgrade of an online cluster with Enterprise Replication

You can perform a rolling upgrade in a high-availability cluster by temporarily converting the primary and secondary servers to stand-alone Enterprise Replication servers. The upgrade occurs without incurring any downtime because Enterprise Replication supports replication between different versions of the server software. You can use this approach to upgrade to a new major version, or to apply fix packs or interim fixes (PIDs).

Before you begin

During this procedure, you convert the primary server and the secondary servers to standalone Enterprise Replication servers. You then upgrade the software on the secondary server, stop Enterprise Replication, and then clone the server using the **ifxclone** command.

See the following link for additional information about upgrading clusters: <http://www.ibm.com/developerworks/data/library/techarticle/dm-1012rollingupgrade/index.dital>.

The following prerequisites apply when upgrading software on a cluster:

- Non-logged databases are not supported.
- Raw or unlogged tables are not supported.
- Typed tables are not supported unless the typed table contains a primary key.
- UDTs that do not support Enterprise Replication are not supported.
- The `CDR_QDATA_SBSPACE` configuration parameter must be set on both the primary and secondary servers.
- The `sqlhosts` file must define a server group.
- The primary and secondary servers must belong to different groups.
- For versions of Informix® software earlier than 11.50.xC7, converting a primary and secondary server pair to Enterprise Replication is not supported if a table does not have a primary key.
- For versions of Informix® software earlier than 11.10, the `sec2er` command is not supported.

If you are upgrading Informix® software version 11.50.xC7 or later, the `sec2er` command adds a primary key to any table that does not already have one defined. For large tables, adding the primary key can take a long time, during which you will not see any server activity. In addition, the `sec2er` command requires exclusive access to the table while adding the primary key and user transactions will be blocked from accessing the table. You might want to manually create primary keys on any large table before running the `sec2er` command. If you have tables that were created with the `DELIMIDENT` environment variable set, and the tables do not have primary keys, then you must manually create the primary keys for those tables before running the `sec2er` command.

About this task

There are different steps involved in the upgrade process depending on whether you are using the Connection Manager:

- For high-availability clusters that use the Connection Manager to redirect user connections:
There are two options you can choose based on your requirements. You can:
 - Add a new Connection Manager instance to manage user connections while the cluster is upgraded. This involves configuring a new 3.70 Connection Manager instance that supports Enterprise Replication and has corresponding changes to the `sqlhosts` file or other connection mechanisms for user applications. If users already have a Connection Manager group support infrastructure to manage their user connections, they can easily add the new Connection Manager for Enterprise Replication to their existing Connection Manager group to ensure that no user connection downtime occurs during the upgrade process.
 - Use your existing cluster Connection Manager or Connection Manager groups throughout the upgrade process, without making any changes to the Connection Manager configuration, applications, or application connection mechanisms. This option has a 10-second down time for user connections, but if that is acceptable, you can avoid the overhead of adding a new Connection Manager instance and the configuration changes that go with it.
- For clusters not using the Connection Manager to redirect user connections:
Users must take steps to move user connections to the appropriate servers during the upgrade process.

Performing a server upgrade when the Connection Manager is not in use

In this example, the terms `server1` and `server2` refer to server names rather than machine names.

Some additional steps are required to upgrade Informix® software:

1. On the primary server (**server1**), perform a check to see whether the servers can be split into Enterprise Replication servers by running the following command:

```
cdr check sec2er -c server1 --print server2
```

The command examines the primary and secondary servers and determines if it is possible to convert them to Enterprise Replication. The command displays warnings and errors that explain conditions that may prevent the servers from converting to Enterprise Replication. The `-print` option prints the commands that will be run when the `cdr start` command runs. You should fix any warnings or errors and then run the command again before performing the next step.

2. Run the following command from an Informix® 11.70 or later server:

```
cdr start sec2er -c server1 server2
```

The `sec2er` command converts the primary and secondary servers into standalone servers and configures and starts Enterprise Replication. Enterprise Replication keeps the data on the servers synchronized; however, any table created after the `sec2er` command is run will not be replicated.

3. On the former secondary server (**server2**), upgrade the Informix® software. The steps to upgrade the server are as follows:

- a. Stop replication by running the following command:

```
cdr stop
```

- b. Back up the logical logs:

```
ontape -a
```

- c. Stop the server that contains the older version of the Informix® software:

```
onmode -kuy
```

- d. Log on to the server with the newly installed Informix® software.

- e. Start the server and let the conversion complete successfully :

```
oninit
```

- f. Run the `concdr.sh` script to convert the `syscdr` database from the old software version to the new version:

```
concdr.sh old_version new_version
```

- g. Start replication on the former secondary server (**server 2**) after it has been upgraded:

```
cdr start
```

Because Enterprise Replication supports replication between dissimilar versions of the server software, the upgraded secondary server (**server2**) replicates data with the former primary server (**server1**), so that data updates are replicated on both servers.

4. Move client application connections from the former primary server (**server1**) to the upgraded server (**server2**).
5. On the primary server (**server1**) use the `onmode -k` command to take the database server to offline mode.

```
onmode -k
```

6. On the former secondary server (**server2**) run the following command to stop Enterprise Replication:

```
cdr stop
```

- You can now clone the upgraded server to set up the other secondary servers in your cluster. Clone the newly upgraded server (**server2**) by running the `ifxclone` utility on **server1**. Use the `-d` (disposition) parameter to create a standalone, RSS, or HDR secondary server. In the following examples, assume that the TCP/IP address for **server1** is 192.168.0.1 on port 123, and the address for **server2** is 192.168.0.2 on port 456.

- To create a standalone server:

```
ifxclone -T -S server2 -I 192.168.0.2 -P 456 -t server1
        -i 192.168.0.1 -p 123
```

- To create an RS secondary server, specify the disposition by using the `-d RSS` option:

```
ifxclone -T -S server2 -I 192.168.0.2 -P 456 -t server1
        -i 192.168.0.1 -p 123 -d RSS
```

- To create an HDR secondary server, specify the disposition by using the `-d HDR` option:

```
ifxclone -T -S server2 -I 192.168.0.2 -P 456 -t server1
        -i 192.168.0.1 -p 123 -d HDR
```

At this point, the cluster is running on the upgraded server. Clients can move applications from `server2` if necessary.

Performing a server upgrade when the Connection Manager is in use

Refer to the following steps when clients are using the Connection Manager without a Connection Manager group defined in the existing setup.

For this example, assume that the following Connection Manager configuration file is defined:

```
NAME cm1
LOG 1
LOGFILE /tmp/cm1.log

CLUSTER cluster1
{
  INFORMIXSERVER ids1,ids2
  SLA oltp DBSERVERS=primary
  SLA webapp DBSERVERS=HDR
  SLA report DBSERVERS=(primary,HDR)
  FOC ORDER=ENABLED PRIORITY=1
}
```

- On the primary server (**server1**), perform a check to see whether the servers can be split into Enterprise Replication servers by running the following command:

```
cdr check sec2er -c server1 --print server2
```

When the above command is run, the primary and secondary servers are examined to determine whether it is possible to convert them to Enterprise Replication. The command displays warnings and errors that explain conditions that might prevent the servers from converting to Enterprise Replication. The `-print` option prints the commands that will be run when the `cdr start sec2er` command runs. You should fix any warnings or errors and then run the command again before performing the next step.

- Reload the Connection Manager so that it directs all client connections to the primary server. Here is the revised Connection Manager configuration file:

```
NAME cm1
LOG 1
```

```

LOGFILE /tmp/cm1.log

CLUSTER cluster1
{
  INFORMIXSERVER ids1,ids2
  SLA oltp DBSERVERS=primary
  SLA webapp DBSERVERS=primary,HDR
  SLA report DBSERVERS=primary,HDR
  FOC ORDER=ENABLED PRIORITY=1
}

```

3. Run the following command from an Informix® 11.70 or later server:

```
cdr start sec2er -c server1 server2
```

The `sec2er` command converts the primary and secondary servers into standalone servers and configures and starts Enterprise Replication. Enterprise Replication keeps the data on the servers synchronized; however, any table created after the `sec2er` command is run will not be replicated.

4. On the former secondary server (**server2**), upgrade the Informix® software.

Because Enterprise Replication supports replication between dissimilar versions of the server software, the upgraded secondary server (**server2**) replicates data with the former primary server (**server1**), so that data updates are replicated to both servers.

5. Move client application connections from the former primary server (**server1**) to the upgraded server (**server2**).
 - a. Create a new Connection Manager configuration file for Enterprise Replication. The following shows a sample Enterprise Replication Connection Manager configuration file. The SLA names are same as for `cm1`:

```

NAME cm2
LOG 1
LOGFILE /tmp/cm2.log
MACRO list=g_server2,g_server1

REPLSET replset_1
{
  INFORMIXSERVER g_server1,g_server2
  SLA oltp DBSERVERS=${list}
  SLA webapp DBSERVERS=${list}
  SLA report DBSERVERS=${list}
}

```

The Enterprise Replication Connection Manager must define a replicate set that includes all replicates that are generated by the `sec2er` command. You can see the list of replicates by running the following command:

```
cdr list repl
```

You create a replicate set by running the following command:

```
cdr def replset replset_name repl1 repl2 ...
```

In the above example, `repl1` and `repl2` are replicates created by the `sec2er` command.

- b. Halt the **cm1** Connection Manager instance and load the `cm2` instance.

Performing the above step ensures that client connections are redirected to **group_2** (because **server2** belongs to **group_2**).

Here is a sample `sqlhosts` file:

```
#dbservername  nettype  hostname  servicename  options
g_server1     group    -         -            i=10
ids1          onsoctcp host1     port1       g=g_server1
g_server2     group    -         -            i=20
ids2          onsoctcp host2     port2       g=g_server2

oltp          onsoctcp host1     port3
webapp        onsoctcp host1     port4
report        onsoctcp host1     port5
```

6. On the primary server (**server1**) use the `onmode -k` command to take the database server to offline mode.

```
onmode -k
```

7. You can now clone the upgraded server to the other secondary servers in your cluster.

At this point in the upgrade process, the high-availability cluster is running on the upgraded server.

8. Shut down the `cm2` Connection Manager instance and start the `cm1` instance.
9. On the former secondary server (**server2**) run the following command to stop Enterprise Replication:

```
cdr stop
```

Errors and warnings generated by the `sec2er` command

The `sec2er` command checks several conditions before converting a primary and secondary server pair to an ER system. The following conditions are checked by the `sec2er` command; ER conversion will take place only if the following conditions are met:

- The group definition must use the `i=` option.
- The CDRID option must be the same on both the primary and secondary servers (The CDRID is the unique identifier for the database server in the Options field of the `sqlhosts` file).
- The `sqlhosts` files on the primary server must match the `sqlhosts` file on the secondary server. The `sec2er` command checks only the lines in the `sqlhosts` files that must to match to support ER.
- The database must not contain a typed table without a primary key.
- User-defined types (UDT) must have ER support.
- Tables must not be protected with label-based access control (LBAC).
- A secondary server must be defined.
- An sbspace for a stable queue must exist.
- You must be running Informix® version 11.00 or later.

The following warnings might occur. Warnings do not always indicate a problem but should be addressed. A warning is generated if any of the following are true:

- The `CDR_SERIAL` configuration parameter is not set.
- The values for the `CDR_SERIAL` configuration parameter are the same on both the primary and secondary servers. Identical values can cause conflicts.

- The database has sequence generators. Because sequence generators are not replicated, if you replicate tables using sequence objects for update, insert, or delete operations, the same sequence values might be generated on different servers at the same time, leading to conflicts.
- The database is not logged.
- A table is not logged.
- The DBSPACE is more than 70-percent full.

If the `cdr start sec2er` command fails or is interrupted, you might see a message that is similar to this message:

```
ERROR: Command cannot be run on pre-11.70 instance if ER is already running
```

If you receive this error, and you do not have the Connection Manager running, remove replication by running the `cdr delete server` command on both servers and then run the `cdr start sec2er` command again. If you have the Connection Manager running, then perform the following steps:

1. Shut down the Connection Manager.
2. Shut down ER using one of the following commands:
 - `cdr delete server group`
 - `cdr delete server server`
3. Start the Connection Manager.
4. Restart the **sec2er** command:

```
cdr start sec2er
```

Reverting clusters

If you have a high-availability cluster, you must complete additional tasks when you revert from the new version of Informix®. You must revert only the primary database server.

About this task

The server automatically removes secondary servers during reversion. After reversion on the primary server is complete, you must recreate all HDR, RS, and SD secondary servers in a high-availability cluster.

Prerequisites:

- Determine if you can revert. See information in [Review the database schema prior to reversion on page 52](#).
- Complete the steps in [Preparing to migrate, upgrade, or revert clusters on page 24](#).
- Perform all reversion operations as user **informix**.

When you revert clusters, be sure to stop and start the servers in the cluster in the order shown in the following procedure.

To revert high-availability clusters:

1. Stop the Connection Manager by issuing the `oncmism -k connection_manager_name` command.
2. If you are using a High-availability Data Replication (HDR) secondary server as a backup server in case of contingencies:
 - a. Quiesce the primary server by issuing an `onmode -sy` command to prevent user connections to the server.
 - b. Force a checkpoint by issuing an `onmode -c` command on the primary server.
3. Stop the servers in the cluster and perform the following tasks in the following order:
 - a. If you have remote standalone (RS) servers, stop them by issuing the `onmode -ky` command.
 - b. If you have shared disk (SD) servers, stop them by issuing the `onmode -ky` command.
 - c. If you have a High-availability Data Replication (HDR) secondary server, stop it by issuing the `onmode -ky` command.
 - d. Revert the standard server by issuing an `onmode -b target_IDS_version` command.
 - e. Verify that reversion was successful and the server was stopped. If the reversion was not successful, check the message log for error messages, take appropriate action, and restart reversion.
4. On each server, set the `INFORMIXDIR` environment variable to the full path name for the target installation.
5. Ensure that all of the necessary configuration files are available in the target installation.
6. Perform any additional database server reversion tasks, as described in [Reverting from Informix Version 14.10 on page 57](#).
7. Start the primary server by issuing an `oninit` command.
8. Start the Connection Manager by issuing an `oncmism` command.
9. Start SD secondary servers by issuing `oninit` commands.
10. Back up all logs. Then use ON-Bar or **ontape** to make a level-0 backup on the primary server to use to reestablish the RS and HDR servers if necessary.
11. If you have RS secondary servers:
 - a. Add RS entries on the primary server by issuing `onmode -d add RSS rss_server_name` commands.
 - b. Start the RS secondary servers with level-0 restore operations from the level 0 backup that was made on the primary server after reversion.
 - c. On the RS secondary servers, run the `onmode -d RSS primary_server_name` command, and wait for the "`RSS secondary server operational`" message to appear after each command.
12. If you have an HDR secondary server:
 - a. Reestablish the HDR pair on the primary server by issuing an `onmode -d primary_hdr_secondary_server_name` command.
 - b. Start the HDR secondary server with level-0 restore operations from the level 0 backup that was made on the primary server after reversion.
 - c. On the HDR secondary server, run the `onmode -d secondary primary_server_name` command, and wait for the "`HDR secondary server operational`" message to appear after each command.

Results

The reversion of all servers in the cluster is now complete.

Related information[Preparing to revert on page 51](#)

Restoring clusters to a consistent point

You can restore the primary server in a high-availability cluster to a consistent point after a failed upgrade.

Before you begin

Prerequisites:

- Before you began the upgrade, you must have enabled quick reversion, according to information in [Configuring for recovery of restore point data in case an upgrade fails on page 10](#).
- The server must be offline.

About this task

To restore the primary server in a cluster to a consistent point after a failed upgrade:

Run the **onrestorept** utility. For more information, see [Restoring to a previous consistent state after a failed upgrade on page 45](#).

What to do next

Alternatively, if you backed up your primary server or you prepared for using the High-availability Data Replication (HDR) secondary server as a backup server before you upgraded and the upgrade fails, you can take other steps to restore the cluster. See [Restoring a cluster from a backup archive on page 39](#) or [Restoring a cluster from the HDR secondary server on page 40](#).

Restoring a cluster from a backup archive

If you backed up the primary server before you migrated or reverted the cluster, you can restore the primary server from the backup archive if migration or reversion fails. After you restore the primary server, you must recreate the other servers in the high-availability cluster.

Before you begin

Prerequisite: You made a level-0 backup archive of the primary server before migration or reversion.

About this task

To restore a cluster from a level-0 backup archive:

1. Point your INFORMIXDIR, PATH, and any other relevant environment variables to the directory in which the original version of Informix® was installed before you migrated or reverted.
2. Using the level-0 backup archive, perform a full restore of your primary server.
3. Recreate the rest of your high-availability cluster.

Restoring a cluster from the HDR secondary server

You can restore the primary server from the High-availability Data Replication (HDR) secondary server that you prepared to use as a backup server before you migrated or reverted the cluster. After you restore the primary server, you must recreate the other servers in the high-availability cluster.

Before you begin

Prerequisite: You prepared the HDR secondary server to use as a contingency backup server, according to information in [Preparing to migrate, upgrade, or revert clusters on page 24](#).

To restore a cluster from the HDR secondary server:

1. Start the HDR secondary server by running an oninit command.
2. Change the secondary server to the primary server by running the onmode -d make primary *hdr_server_name* command.
3. If the server is in quiescent mode, change it to multi-user mode by running an onmode -m command.
4. Make a level-0 backup using the ON-Bar or **ontape** utility.
5. Recreate the rest of the high-availability cluster.

Migrating to Informix® 14.10

When you migrate to a new version of Informix®, you must complete required migration and post-migration tasks.

Migrating to the new version of Informix®

After you prepare your databases for migration, you can migrate to the new version of Informix®.

Before you begin

- Read the release notes and the machine notes for any new information.
- Complete the steps in [Preparing for migration on page 7](#).

About this task

To upgrade a Informix® non-root installation, you must run the installation program as the same user who installed the product being upgraded.

If you are migrating the database server from a version that does not support label-based access control, users who held the DBA privilege are automatically granted the SETSESSIONAUTH access privilege for PUBLIC during the migration process. For more information about SETSESSIONAUTH, see the *Informix® Guide to SQL: Syntax*. For information about label-based access control, see the *Informix® Security Guide*.



Important: Do not connect applications to a database server instance until migration has successfully completed.

Review and complete all tasks that apply:

1. [Installing the new version of Informix on page 41.](#)
2. [Setting environment variables on page 42.](#)
3. [Customizing configuration files on page 42.](#)
4. [Adding Communications Support Modules on page 44.](#)
5. [Installing or upgrading any DataBlade modules on page 44.](#)
6. [Starting the new version of Informix on page 44.](#)

Results

When the migration starts, the `online.log` displays the message `Conversion from version <version number> Started..` The log continues to display start and end messages for all components. When the migration of all components is complete, the message `Conversion Completed Successfully` appears. For more information about this log, see [Migration status messages on page 42.](#)

What to do next

After migration, see [Completing required post-migration tasks on page 46](#) for information about preparing the new server for use.

If the log indicates that migration failed, you can either:

- Install the old database server and restore your database from a level-0 backup.
- Run the `onrestorept` utility to back out of the upgrade and restore files to a consistent state without having to restore from a backup. You can run this utility only if you set the configuration parameters that enable the utility. See [Restoring to a previous consistent state after a failed upgrade on page 45.](#)

Installing the new version of Informix®

Install and configure the new version of Informix®.

If possible, migrate on a database server dedicated to testing your migration before you migrate on your production database server.

Decide whether to upgrade on the same computer, known as an in-place migration, or on a different computer, known as a non-in-place migration and follow the appropriate process.



Important: Monitor the database server message log, `online.log`, for any error messages. If you see an error message, solve the problem before you continue the migration procedure.

Related information

Upgrading Informix (in-place migration)

Migrating Informix (non-in-place migration)

Migration status messages

When the migration starts, the **online.log** displays the message "Conversion from version <version number> Started." The log continues to display start and end messages for all components.

When conversions of all components are complete, the message "Conversion Completed Successfully" displays. This message indicates that the migration process completed successfully, but it does not guarantee that each individual database was converted successfully. The message log might contain more information about the success or failure of the migration of each individual database. If migration of a particular database fails, then try to connect to the database to find out the exact cause of the failure.

At the end of the migration of each individual database, Informix® runs a script to update some system catalog table entries. The message log includes messages that are related to this script. The success or failure of the script does not prevent the usage of a database.

For information about any messages in the message log, see the *Informix® Administrator's Guide*.

Setting environment variables

After you install the current version of Informix®, verify that the **INFORMIXDIR**, **INFORMIXSERVER**, **ONCONFIG**, **PATH**, and **INFORMIXSQLHOSTS** (if used) environment variables are set to the correct values.

The client application looks for the **sqlhosts** file in the **etc** directory in the **INFORMIXDIR** directory. However, you can use the **INFORMIXSQLHOSTS** environment variable to change the location or name of the **sqlhosts** file.

The setting of the **GL_USEGLU** environment variable must match between the source and target server during migration.

For information about environment variables, see the *Informix® Guide to SQL: Reference*.

Customizing configuration files

When you initialize the new version of Informix®, which contains a new `onconfig.std` file, use the same configuration that the old database server used. After you observe the performance of new version, you can examine the new file for

new configuration parameters that you might want to use and can you start to use the new and changed configuration parameters.

Set the ALARMPROGRAM configuration parameter to either nothing or **no_log.sh** to prevent the generation of errors if the logical log fills during the migration. For more details, see [Starting the new version of Informix on page 44](#). After the migration, change the value of ALARMPROGRAM to **log_full.sh**.

If the ALARMPROGRAM configuration parameter is set to the script **alarmprogram.sh**, set the value of BACKUPLPGS in **alarmprogram.sh** to `N`.



Important: To facilitate migration (and reversion), use the same values for your new database server for ROOTOFFSET , ROOTSIZE, and ROOTPATH that you used for the old database server. Also, keep the same size for physical logs and logical logs, including the same number of logical logs, and the same **sqlhosts** file.

For information about how to configure Informix®, see your *Informix® Administrator's Guide*. For information about how to tune the configuration parameters, see the *Informix® Performance Guide*.

Using SSL/TLS database connections

If you are upgrading database clients that uses SSL/TLS connections to Client SDK 4.50.xC4W1 or newer, you may need to migrate their client keystores. For more information, see [Configuring a client for SSL connections on page 44](#).

To perform keystore migration:

1. If your database client installation is co-located with the database server installation, the database client continues to use GSKit as encryption library. In this case, keystore migration is not necessary.
2. If your database client uses a stand-alone installation of Client SDK 4.50.xC4W1 or newer, then it will now use OpenSSL as encryption library, rather than GSKit. In this case:

- a. Ensure to have an appropriate version of OpenSSL installed before you install Client SDK 4.50.xC4W1 or newer.
- b. If your client keystore has the GSKit-proprietary format "CMS" (file extension "*.kdb"), then this keystore needs to be converted to a **PKCS#12** keystore. As the CMS format is GSKit-specific, you need the GSKit command "gsk8capicmd" (or "gsk7capicmd") in order to convert the keystore.

Use a command like:

```
gsk8capicmd -keydb -convert -db KEYSTOREFILE.kdb -pw PASSWORD
-old_format cms -new_db KEYSTOREFILE.p12 -new_pw PASSWORD
-new_format pkcs12
```

- c. Create a stash file with the keystore password to use with OpenSSL. Use the new utility "onkstash" contained with Client SDK 4.50.xC4W1 (or newer) to stash the keystore password:

```
onkstash KEYSTOREFILE.p12 PASSWORD
```



Note: This step is also needed in case your keystore already had the PKCS#12 format.

Adding Communications Support Modules

For communications with clients, you can optionally use a Communications Support Module (CSM) with the current version of Informix®. After you install the CSM components, create entries in the **concsm.cfg** file and in the options field of the **sqlhosts** file to configure the CSM.

Existing client applications do not need to be recompiled or relinked if your database server does not use CSMs. If your database server uses a CSM, client applications must relink with new Informix® libraries. The client applications must install and configure the CSM.

For information about how to set up the CSM, see the *Informix® Administrator's Guide*.



Note: Support for Communication Support Module (CSM) is removed starting Informix Server 14.10.xC9 . You should use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) instead.

Installing or upgrading any DataBlade® modules

After you install the new version of Informix®, you might need to install or upgrade any DataBlade® modules that you want to add to the database server.

Register the DataBlade® modules after you initialize the database server.

When you install Informix®, the built-in extensions, such as TimeSeries, are installed and registered automatically. You do not need to perform any actions to upgrade these DataBlade® modules, nor do you need to unload and load any data during migration.



Important: If the sysadmin database does not exist or the Scheduler is turned off, automatic registration does not occur. You must register each extension that you want to use by running the SYSBldPrepare() function.

Starting the new version of Informix®

After installing the new database server, start the server. Do not perform disk-space initialization, which overwrites whatever is on the disk space.

Prerequisite: If you installed Informix® as user **root**, you must switch to user **informix** before starting the server.



Important: Informix® writes to the logical logs with the transactions that result from creating the **sysmaster** database. If you run out of log space before the creation of the **sysmaster** database is complete, Informix® stops and indicates that you must back up the logical logs. After you back up the logical logs, the database server can finish building the **sysmaster** database. You cannot use ON-Bar to back up the logical logs because the database has not been converted yet. If you have ALARMPROGRAM set to **log_full.sh** in the ONCONFIG configuration file, errors are generated as each log file fills during the migration. Set the value of ALARMPROGRAM to either nothing or **no_log.sh** so that these errors are not generated. If your logical log does fill up during the migration, you must back it up with **ontape**, the only backup tool you can use at this point. Issue the **ontape -a** command.

Start the new version of Informix® for the first time by running **oninit** command on UNIX™ or by using the **Service** control application on Windows™.

As Informix® starts for the first time, it modifies certain disk structures. This operation should extend the initialization process by only a minute or two. If your disks cannot accommodate the growth in disk structures, you will find a message in the message-log file that instructs you to run an **oncheck** command on a table. The **oncheck** utility will tell you that you need to rebuild an index. You must rebuild the index as instructed.

Restoring to a previous consistent state after a failed upgrade

If the **CONVERSION_GUARD** configuration parameter is enabled and an upgrade fails, you can run the **onrestorept** command to undo the changes that are made during the upgrade and restore Informix® to a consistent state.

Before you begin

The following prerequisites must be met:

- The directory that is specified by the **RESTORE_POINT_DIR** configuration parameter contains the files that were stored during the failed upgrade attempt.
- The server must be offline before you run the **onrestorept -y** command.

About this task

To restore the server to a previous consistent state after a failed upgrade:

Run the **onrestorept -y** command, which displays prompts while the command runs. If you do not specify **-y**, you must respond to every prompt that appears.



Important: Do not start the server until the **onrestorept** utility finishes running. Starting the server before the **onrestorept** utility finishes running can damage the server, requiring the server to be restored from a backup copy.

What to do next

After you restore the server to a consistent state, you can resume work in the source version of the server or find and fix the problem that caused the upgrade to fail.



Important: To start Enterprise Replication after the onrestorept utility restores the database server to a consistent state, you must use the cdr cleanstart command.

Before you attempt another upgrade, run the onrestorept -c command to remove the files in the directory that is specified in the RESTORE_POINT_DIR configuration parameter. After a successful upgrade, the server automatically deletes restore point files from that directory.

Related information

[The onrestorept utility on page 131](#)

[Preparing for migration on page 7](#)

Completing required post-migration tasks

After you migrate, you must complete a series of post-migration tasks to prepare the new version of the server for use.

To complete post-migration tasks:

1. If you **do not** use Informix® Primary Storage Manager: [For ON-Bar, copy the sm_versions file on page 47.](#)
2. [Finish preparing earlier versions of 12.10 databases for JSON compatibility on page 47](#)
3. [Optionally update statistics on your tables after migrating on page 48.](#)
4. [Review client applications and registry keys on page 48.](#)
5. [Verify the integrity of migrated data on page 49.](#)
6. [Back up Informix after migrating to the new version on page 50.](#)
7. [Tune the new version for performance and adjust queries on page 50.](#)
8. If you use specific features, you might have to perform additional post-migration tasks:
 - [Migrating with Enterprise Replication on page 18](#)
 - [High-availability cluster migration on page 21](#)
 - [Register DataBlade modules on page 50](#)

What to do next

Repeat the migration and post-migration procedures for each instance of Informix® Version 14.10 that you plan to run on the computer.



Important: Do not connect applications to a database server instance until the migration has completed successfully. If a serious error occurs during the migration, you might need to revert to the previous version of the server, restore from a level-0 backup, and then correct the problem before restarting the migration tasks.

Related information

[Reverting from Informix Version 14.10 on page 51](#)

For ON-Bar, copy the sm_versions file

After migration, if you plan to use a storage manager other than the Informix® Primary Storage Manager, copy your previous `sm_versions` file to use with ON-Bar. Informix® Primary Storage Manager does not use the `sm_versions` file.

If you are using other storage managers, copy your previous `sm_versions` file from the old `$INFORMIXDIR/etc` directory to the new `$INFORMIXDIR/etc` directory.

Finish preparing earlier versions of 12.10 databases for JSON compatibility

To make databases that were created with earlier versions of Informix® 12.10 JSON compatible, you must complete some post-migration steps.

1. Prepare any databases that were created in 12.10.xC1 for JSON compatibility.

If you upgraded to 12.10.xC4 or later fixpacks, skip this step because all databases are made JSON compatible during conversion.

If you upgraded to 12.10.xC3 or 12.10.xC2, complete these steps:

- a. Run the appropriate script on the upgraded server as user **informix** or as a user with DBA privileges.

- Informix® 12.10.xC3: `convTovNoSQL1210.sql`
- Informix® 12.10.xC2: `convTovNoSQL1210X2.sql`

Example

For example, in Informix® 12.10.xC3, to make the **db_name** database JSON compatible, you would run the following command as user **informix** or as a user with DBA privileges:

UNIX™

```
dbaccess -e db_name $INFORMIXDIR/etc/convTovNoSQL1210.sql
```

Windows™

```
dbaccess -e db_name %INFORMIXDIR%\etc\convTovNoSQL1210.sql
```

- b. Configure and start the wire listener.
2. If you used the wire listener in 12.10.xC2 or 12.10.xC3, after you upgrade to 12.10.xC4 or later fixpacks you must drop and recreate any indexes that you created on your collections. You do not need to drop and recreate the index that is automatically created on the `_id` field of a collection.
 3. If you had binary JSON (BSON) DATE fields in your documents in 12.10.xC2, you must load the data from the external table that you created before you upgraded to a later 12.10 fix pack.

On the upgraded server:

- a. Rename the table that contains the BSON column with DATE fields.

For example, use the ALTER table statement to rename the table from **datetab** to **datetab_original**.

- b. Create a new table that has the original table name.

For example:

```
create table datetab(j int, i bson);
```

- c. Load data into the new table from the external table that you created in your 12.10.xC2 server. During the load, convert the data from JSON format to BSON format.

For example:

```
insert into datetab select j, i::json::bson from ext_datetab;
```

- d. Verify that the data loaded successfully.
- e. Drop the original, renamed table (for example, **datetab_original**) after you are certain that the data loaded successfully into the new table (for example, **datetab**).
4. If you used the wire listener in 12.10.xC2 with a database that has any uppercase letters in its name, after you upgrade to a later fix pack you must update your applications to use only lowercase letters in the database name.

Related information

[Preparing 12.10 BSON columns with DATE fields for upgrade on page 12](#)

Optionally update statistics on your tables after migrating

Optionally run UPDATE STATISTICS on your tables and on UDRs that perform queries, if you have performance problems after migrating to the new version of Informix®.

An unqualified UPDATE STATISTICS statement includes no additional clauses:

```
UPDATE STATISTICS;
```

By default, an unqualified UPDATE STATISTICS statement updates the statistics in LOW mode for every permanent table in the database, including the system catalog tables.

You can run an UPDATE STATISTICS statement that includes only a FOR ROUTINE clause that specifies no routine name:

```
UPDATE STATISTICS FOR ROUTINE;
```

Running this statement reoptimizes the DML statement execution plans for every SPL routine in the database that operates on local tables.

Similarly, you can substitute the keyword FUNCTION for ROUTINE in the previous example to reoptimize execution plans only for SPL routines that return at least one value, or you can substitute the keyword FUNCTION for PROCEDURE to reoptimize execution plans only for SPL routines that return no value. In these cases, the database server does not update the statistics in the system catalog tables.

You do not need to run UPDATE STATISTICS statements on C or Java™ UDRs.

Review client applications and registry keys

After you migrate a database server on the same operating system or move the database server to another compatible computer, review the client applications and `sqlhosts` file or registry-key connections. If necessary, recompile or modify client applications.

Verify that the client-application version you use is compatible with your database server version. If necessary, update the `sqlhosts` file or registry key for the client applications with the new database server information.

If you have a 64-bit ODBC application that was compiled and linked with a version of HCL Informix® Client Software Development Kit (Client SDK) that is prior to version 4.10, you must recompile the application after migrating. The `SQLLEN` and `SQLULEN` data types were changed to match the Microsoft™ 64-bit ODBC specification. Be sure to analyze any functions that take either of these data types to ensure that the correct type passes to the function. This step is crucial if the type is a pointer. Also note that in the ODBC specification, some parameters that were previously `SQLINTEGER` and `SQLUINTEGER` were changed to `SQLLEN` or `SQLULEN`.

For more information about interactions between client applications and different database servers, refer to a client manual.

Verify the integrity of migrated data

Open each database with DB-Access and use `oncheck` to verify that data was not corrupted during the migration process.

You can also verify the integrity of the reserve pages, extents, system catalog tables, data, indexes, and smart large objects, as [Table 6: Commands for verifying the data integrity on page 49](#) shows.

Table 6. Commands for verifying the data integrity

Action	oncheck Command
Check reserve pages	<code>oncheck -cr</code>
Check extents	<code>oncheck -ce</code>
Check system catalog tables	<code>oncheck -cc <i>database_name</i></code>
Check data	<code>oncheck -cD <i>database_name</i></code>
Check indexes	<code>oncheck -cl <i>database_name</i></code>
Check smart large objects	<code>oncheck -cs <i>sbspace_name</i></code>
Check smart large objects plus extents	<code>oncheck -cS <i>sbspace_name</i></code>

If the `oncheck` utility finds any problems, the utility prompts you to respond to corrective action that it can perform. If you respond `Yes` to the suggested corrective action, run the `oncheck` command again to make sure the problem has been fixed.

The `oncheck` utility cannot fix data that has become corrupt. If the `oncheck` utility is unable to fix a corruption problem, you might need to contact Software Support before you proceed.



Important: If the value of the `MAX_FILL_PAGES` configuration parameter is 1, you must run `oncheck -cD` for all tables that include variable length data types (`VARCHAR`, `NVARCHAR`, and `LVARCHAR`), and take the suggested corrective action to avoid warnings about resetting the bitmap mode.

Back up Informix® after migrating to the new version

Use a backup and restore tool (`ON-Bar` or **ontape**) to make a level-0 backup of the new database server. Do not overwrite the tapes that contain the final backup of the old database server.

For more information, see the *Informix® Backup and Restore Guide*.



Important: Do not restore the backed up logical-log files from your old database server for your new database server.

Tune the new version for performance and adjust queries

After backing up the new server, you can tune the database server to maximize performance. If your queries are slower after the upgrade, there are steps you can take to adjust your configuration and queries.

If your queries are slower after an upgrade, find out what changed that affects your configuration and adjust your configuration and queries as necessary:

- Compare the default values in the new `onconfig.std` file to values in your previous installation, and make any necessary adjustments.
- If you created sample queries for comparison, use them to analyze the performance differences between the old and new database servers and to determine if you need to adjust any configuration parameters or the layout of databases, tables, and chunks.
- If you changed your applications, check to see if the changes led to slower performance.
- If you changed your hardware, operating system, or network settings, determine if you need to adjust any related settings or environment variables.
- Make sure that you ran necessary update statistics after upgrading:
 - [Drop data distributions if necessary when upgrading on page](#)
 - [Optionally update statistics on your tables after migrating on page 48](#)
 -

Register DataBlade® modules

You must register any DataBlade® modules that you installed.

Registration is the process that makes the DataBlade® module code available to use in a particular database. For more information on how to use DataBlade® modules, see the *Informix® DataBlade® Module Installation and Registration Guide*.

Reverting from Informix® Version 14.10

You can revert to the version of the database server from which you migrated. When you run the reversion utility, you specify the target server for reversion and then Informix® checks your database. If necessary, Informix® might tell you to drop new objects, before automatically converting your data into the target server.

If Informix® cannot revert a database, Informix® prevents reversion.

Normally, reversion takes only a few minutes.

If you used the new features of Version 14.10, reversion time is longer, because you must prepare your database and data for reversion, and you must remove the features that are not supported in the earlier version of the server. The more work you complete in the new version, the more time consuming the reversion. See [Preparing to revert on page 51](#) before you revert.

If you did not use any of the new features of Version 14.10 and you did not complete much work using the new server, you can run the reversion utility and modify the values of the configuration parameters. See [Reverting from Informix Version 14.10 on page 57](#).

Related information

[Completing required post-migration tasks on page 46](#)

Preparing to revert

If you used Version 14.10, you must prepare your system for reversion to the pre-migration version of the database server.

Review and complete all tasks that apply:

1. [Review the database schema prior to reversion on page 52](#).
2. [Check and configure available space for reversion on page 53](#).
3. [Save copies of the current configuration files on page 54](#).
4. [Save system catalog information on page 54](#).
5. [Verify the integrity of the Version 14.10 data on page 54](#).
6. [Back up Informix Version 14.10 on page 55](#).
7. Export or save your data.
8. To prevent reversion issues:
 - a. [Resolve outstanding in-place alter operations on page 55](#).
 - b. If you have empty tables with no extents, drop those tables.
 - c. Defragment partitions by using the SQL administration API.

9. [Remove unsupported features on page 57.](#)
10. [Remove new BladeManager extensions on page 57.](#)

Results

What to do next

If you use high-availability clusters or Enterprise Replication, you must complete extra tasks before you can revert to the pre-migration version of Informix®.

Related information

[Reverting clusters on page 37](#)

[Reverting with Enterprise Replication on page 20](#)

[Reverting from Informix Version 14.10 on page 57](#)

Review the database schema prior to reversion

You must review your database schema to determine if reversion is possible. You can revert from Informix® Version 14.10 to the database server from which you migrated, if you have not added any extensions to the Version 14.10 database server and you are not reverting from a newly created instance.

To review the database schema to determine if reversion is possible:

1. Run the dbschema utility command.

For example, run the following command to display information about the database **db1**:

```
dbschema -d db1 -ss
```

2. Determine if the schema file contains SQL statements that the earlier database server does not support.
3. Determine if the database contains features, such as long identifiers, that the earlier database server does not support.
4. Determine if any new SPL routines have been created in Informix® Version 14.10 or if any routines were imported using dbimport.
5. Determine if tables or indexes using expression fragmentation had expressions changed or new fragments added.
6. Identify any new triggers, procedures, or check constraints.

See [Reversion requirements and limitations on page 52](#) for limitations on reversion to previous databases and prerequisite steps you must take before you revert.



Reversion requirements and limitations

If you used the new database server, you must review a list of reversion requirements and limitations, and then complete any prerequisite tasks before you revert. If the reversion restrictions indicate that you must drop objects from the database, you can unload your data and then reload it in the prior database server.

Reversion requirements and limitations are described in the following tables:

- [Table 7: Requirements and limitations when reverting to any version of the server on page 53](#) Requirements and limitations when reverting to any version of the server

Table 7. Requirements and limitations when reverting to any version of the server

Reversion requirement or limitation
<p>Revert only to the version from which you migrated: If you need to revert, you must revert to the Informix® version that was your source version before you migrated to Version 14.10.</p>
<p>New databases created in the new version of the server: If you created a new database in the new version of the server, you cannot revert the database back to an earlier version of the server. If the data is required, you can unload the data and reload it in the prior version of the server.</p>
<p>New procedures, expression-based fragmented tables, expression-based fragmented indexes, check constraints, and triggers: These cannot be reverted. You must remove any new procedures, fragmented tables, expression-based fragmented indexes, check constraints, and triggers.</p> <p> Note: Expression-based fragmentation includes fragment by expression, fragment by interval, and fragment by list.</p>
<p>New built-in routines: These cannot be reverted.</p>
<p>New configuration parameters or configuration parameters with new options: These cannot be reverted.</p>
<p>New or outstanding in-place alters: In-place ALTER TABLE statements performed in the new version of the server must not be outstanding against any table.</p> <p>Ensure that all in-place ALTER operations are complete. If the reversion process does not complete successfully because of in-place ALTER operations, the reversion process lists all the tables that have outstanding in-place alter operations. You must resolve outstanding in-place alter operations on each of the tables in the list before you can revert to the older database server. For more information, see Resolve outstanding in-place alter operations on page 55.</p> <p> Important: Any in-place alter operation that was completed in a version that is before the current version will successfully revert.</p>

Check and configure available space for reversion

You must be sure you have enough space for reversion to the source database server.

The `tblspace` **tblspace** pages can be allocated in non-root chunks. If the root chunk is full and `tblspace` **tblspace** pages were allocated in non-root chunks, make sure you have enough space in the root chunk of the target database server.

To determine how many pages were allocated and where they were allocated, run **oncheck -pe** and look for the word `TBLSpace`. This space must be available on the device where the root chunk will be located.

For information about space requirements for Informix® Version 14.10, see [Checking and configuring available space on page 8](#).

Related information

[Checking and configuring available space on page 8](#)

Save copies of the current configuration files

Save copies of the `ONCONFIG` and **concsm.cfg** files for when you migrate to Informix® Version 14.10 again.

Informix® uses the **concsm.cfg** file to configure CSMs.

Save system catalog information

If your current database server instance uses secure-auditing masks or external spaces, and you want to preserve the associated catalog information, you must unload these system catalog tables before you revert.

Run the following command to unload the system catalog tables:

```
$INFORMIXDIR/etc/smi_unld
```

When the **smi_unld** utility finishes unloading the information, the utility displays instructions for reloading the information. Save these instructions. After you complete the reversion and bring up your database server, you can reload the data that you preserved. Follow the instructions given with the **smi_unld** utility for reloading the information. Typically, you run the following command:

```
$INFORMIXDIR/etc/smi_load $INFORMIXDIR/etc/
```

Verify the integrity of the Version 14.10 data

Verify the integrity of your Version 14.10 data, if you did not do this after you migrated.

To verify the integrity of your data, run the following commands:

```
oncheck -cI database_name
oncheck -cD database_name
oncheck -cr
oncheck -cc database_name
```


If the **oncheck** utility finds any problems, the utility prompts you to respond to corrective action that it can perform. If you respond **Yes** to the suggested corrective action, run the **oncheck** command again to make sure the problem has been fixed.

The **oncheck** utility cannot fix data that has become corrupt. If the **oncheck** utility is unable to fix a corruption problem, you might need to contact Technical Support before you proceed.

You will also need to verify the integrity of your data after you revert.

Back up Informix® Version 14.10

Before you begin the reversion, make a complete level-0 backup of Informix® Version 14.10.

For more information, see the *Informix® Backup and Restore Guide*.

Resolve outstanding in-place alter operations

Resolving outstanding in-place alter operations is not a requirement before *converting* to a higher database server version. However, if it becomes necessary to *revert* to a previous version you must resolve new outstanding in-place alter operations first.

An in-place alter operation is *outstanding* when data pages still exist with the prior definition, a state that can be detected with the `oncheck -pT` command. An in-place alter operation is *new* if the ALTER statement is executed in the higher database version. Carryovers—outstanding in-place alter operations that existed prior to a conversion—need not be resolved before a subsequent reversion to the earlier server version.

If the reversion process detects new outstanding in-place alter operations, reversion fails and the message log will contain a list of all tables whose in-place alter operations must be resolved before the reversion will succeed.

If you are reverting from version 12.10.xC4 or later, you can remove in-place alter operations by running the `admin()` or `task()` SQL administration command with the `table update_ipa` or `fragment update_ipa` argument. You can include the `parallel` option to run the operation in parallel. For example, the following statement removes in-place alter operations in parallel from a table that is named **auto**:

```
EXECUTE FUNCTION task('table update_ipa parallel','auto');
```

If you are reverting from an earlier version of 12.10, you can resolve outstanding in-place alter operations by running sample UPDATE statements. Sample UPDATE statements force any outstanding in-place alter operations to complete by updating the rows in the affected tables. To generate a sample UPDATE statement, create an UPDATE statement in which a column in the table is set to its own value. This forces the row to be updated to the latest schema without changing column values. Because the database server always alters rows to the latest schema, a single pass through the table that updates all rows completes all outstanding in-place alter operations.

The sample UPDATE statement differs from a standard UPDATE statement because it does not change the data. A standard UPDATE statement usually changes the value of the affected row.

For example, to create a sample update, specify:

```
UPDATE tab1 SET col1=col1 WHERE 1=1 ;
```

You must ensure that the column selected is a numeric data type (for example, INTEGER or SMALLINT) and not a character data type.

If a table is large, a single update of the whole table can cause a long transaction. To avoid a long transaction, update the table in pieces, by ranges of some column, with this statement:

```
... WHERE {id_column} BETWEEN {low_value} AND {step_value}
```

For example, specify:

```
UPDATE tab1 SET col1=col1 WHERE col1 BETWEEN 1 AND 100;
UPDATE tab1 SET col1=col1 WHERE col1 BETWEEN 101 AND 200;
```

Ensure that the UPDATE statements include the entire data set.

If the table is replicated with Enterprise Replication, the database server replicates all updated rows unnecessarily. To avoid replication, update the table as follows:

```
BEGIN WORK WITHOUT REPLICATION;
...
COMMIT WORK;
```

When all the pending in-place alter operations are resolved, run the oncheck -pT command again for each table. In the output of the command, check information in the `Version` section. The number of data pages should match with current version. Also, all other table versions should have `count=0` for the number of data pages that the version is accessing.

For example, if you run the oncheck -pT testdb:tab1 command after outstanding in-place alter operations are resolved, you might see information similar to the information in this segment of sample output:

```
TBLspace Report for testdb:root.tab1

Physical Address      1:860
Creation date        06/23/2011 14:23:08
TBLspace Flags      800801 Page Locking
                    TBLspace use 4 bit bit-maps

Maximum row size     29
Number of special columns  0
Number of keys       0
Number of extents    1
Current serial value  1
Current SERIAL8 value 1
Current BIGSERIAL value 1
Current REFID value  1
Pagesize (k)         2
First extent size    8
Next extent size     8
Number of pages allocated 8
Number of pages used  4
Number of data pages  3
  << Number of data pages used is 3 >>
Number of rows       6
Partition partnum    1048981
Partition lockid     1048981

Extents
  Logical Page  Physical Page  Size Physical Pages
            0            1:1895    8            8

TBLspace Usage Report for testdb:root.tab1
```

```

Type                Pages      Empty  Semi-Full    Full  Very-Full
-----
Free                4
Bit-Map             1
Index               0
Data (Home)         3
-----
Total Pages         8

Unused Space Summary

  Unused data slots                177

Home Data Page Version Summary

Version                Count
-----
  3 (oldest)                0
    << Other version should show data page count=0>>
  4                          0
    << Other version should show data page count=0>>
  5 (current)                3
    << Current should always match the number of data pages>>

```

Remove unsupported features

Before you revert, remove all features that your older database server does not support.



Important: Do not remove objects that you did not create, such as the boot scripts (`boot90.sql` and `boot901.sql`) in the system catalog because the reversion utility uses them.

For a list of features that you must remove before reversion, see [Review the database schema prior to reversion on page 52](#).

Remove new BladeManager extensions

When BladeManager or SQL Registration are used to register an extension in a database, the `ifxmng` Client SDK module, which manages extensions, is registered first. If you need to revert from Version 14.10, and you ran BladeManager against a database, you must remove all BladeManager extensions.

To remove the BladeManager extensions, you must use BladeManager to unregister all Client SDK modules and then run the following BladeManager command:

```
unprep database_name
```

Reverting from Informix® Version 14.10

After preparing to revert, run the reversion utility and prepare to use the original database server.

Before you begin

Prerequisite: Complete the steps in [Preparing to revert on page 51](#). Preparation includes determining if reversion is possible and preparing your database for reversion.

About this task

Review and complete all tasks that apply:

1. [Run the reversion utility on page 58](#).
2. [Restore original configuration parameters on page 59](#).
3. [Restore original environment variables on page 59](#).
4. [Remove any Communications Support Module settings on page 59](#).
5. [Recompile Java user-defined routines on page 60](#).
6. [Reinstall and start the earlier database server on page 60](#).
7. [Optionally update statistics on your tables after reverting on page 61](#).
8. [Verify the integrity of the reverted data on page 61](#).
9. [Back up the database server after reversion on page 61](#).
10. [Return the database server to online mode on page 62](#).
11. If you use high-availability clusters, perform additional tasks that are described in [Reverting clusters on page 37](#).

Results

After reversion to Informix® Version 12 or earlier, the database server automatically drops the **sysadmin** database.



Attention: When you revert to a previous version of the database server, do not reinitialize the database server by using the `-i` command-line parameter. Using the `-i` parameter for reversion would reinitialize the root dbspace, which would destroy your databases.

Related information

[Preparing to revert on page 51](#)

Run the reversion utility

After preparing to revert, run the reversion utility by using an `onmode -b` command.



Important: You must revert to the version of Informix® that was your source database before you migrated. If you revert to a different version of the server, you will corrupt data.

Informix® Version 14.10 must be running when you run the reversion utility. If the reversion utility detects and lists any remaining features that are specific to Informix® Version 14.10, you must remove those features before reversion can complete.

To see a list of all of the versions to which you can revert using an `onmode -b` command, type `onmode -b`.

To run the reversion utility, type

```
onmode -b version_number
```

For examples of the `onmode -b` command, see [Syntax of the onmode -b command on page 130](#).

When you revert to the older version, Informix® displays messages that tell you when reversion begins and ends.

When the reversion is complete, Informix® is offline. The reversion utility drops the Informix® Version 14.10 system catalog tables and restores compatibility so that you can access the data with the earlier version of the database server. The reversion utility does not revert changes made to the layout of the data that do not affect compatibility.

Related information

[Syntax of the onmode -b command on page 130](#)

[Preparation for using the onmode -b command on page 130](#)

[What the onmode -b command does on page 129](#)

Restore original configuration parameters

Replace the Informix® Version 14.10 ONCONFIG configuration file with the ONCONFIG file that you saved before you migrated. Alternatively, you can remove configuration parameters that the earlier database server does not support.

You might also need to adjust the values of existing configuration parameters.

Restore original environment variables

Reset the environment variables to values that are appropriate for the earlier database server.

Remove any Communications Support Module settings

If your Informix® Version 14.10 instance used CSMs, edit the `sqlhosts` file to remove any `csn` option settings that are not supported in the older database server.

If you do not do this, the older database server will return an invalid `sqlhosts` options error.

You must also delete the `concsn.cfg` file if the older database server does not support CSMs.



Note: Support for Communication Support Module (CSM) is removed starting Informix Server 14.10.xC9 . You should use Transport Layer Security (TLS)/Secure Sockets Layer (SSL) instead.

Recompile Java™ user-defined routines

After you revert and before you start the earlier server, recompile Java™ user-defined routines (UDRs) that were compiled with a Java™ development kit version that is earlier than or equal to the version included with the previous server.

What you do depends on whether your application uses external JAR and class files or JAR files installed on the server:

- If your application uses external JAR and class files (for example, JAR and class files that are listed in JVPCLASSPATH), recompile the files.
- If your application uses JAR files installed in the server (for example, through the `install_jar()` support function), you must remove the old JAR file (by using the `remove_jar()` support function) and reinstall the recompiled JAR file in the database.

Reinstall and start the earlier database server

Reinstall and configure the earlier version of the database server.

Refer to the instructions in your *Informix® Installation Guide* and your *Informix® Administrator's Guide*.



Before you start the server: If you have time series data and you reverted to Informix® 12.10.xC1, you must install the patch `patch-idsdb00493404.tar` on Informix® 12.10.xC1. You can obtain the patch from the [Support Portal](#).

Run the `oninit -s` command to start the earlier database server in quiescent mode.



Important: Do **not** use the `oninit -i` command.

Add JSON compatibility to databases that were created in 12.10.xC1

Databases that were created in Informix® 12.10.xC1 are not JSON compatible. In 12.10.xC2 or 12.10.xC3 you can run a script to make those databases JSON compatible. You do not have to run the script after you upgrade to 12.10.xC4 or later fixpacks because all databases are made JSON compatible during conversion.

Before you begin

This procedure requires Informix® 12.10.xC2 or 12.10.xC3.

You must run the script as user **informix** or as a user with DBA privileges.

Run the appropriate script against a database that was created in Informix® 12.10.xC1 to make the database JSON compatible.

Choose from:

- Informix® 12.10.xC3: `convTovNoSQL1210.sql`
- Informix® 12.10.xC2: `convTovNoSQL1210X2.sql`

Example

For example, to make the **db_name** database JSON compatible in Informix® 12.10.xC3, you would run the following command as user **informix** or as a user with DBA privileges:

UNIX™

```
dbaccess -e db_name $INFORMIXDIR/etc/convTovNoSQL1210.sql
```

Windows™

```
dbaccess -e db_name %INFORMIXDIR%\etc\convTovNoSQL1210.sql
```

Optionally update statistics on your tables after reverting

Optionally run UPDATE STATISTICS on your tables and on UDRs that perform queries, if you have performance problems after reverting to the previous version of the database server or to a database server on a different operating system.

An unqualified UPDATE STATISTICS statement includes no additional clauses:

```
UPDATE STATISTICS;
```

By default, an unqualified UPDATE STATISTICS statement updates the statistics in LOW mode for every permanent table in the database, including the system catalog tables.

You can run an UPDATE STATISTICS statement that includes only a FOR ROUTINE clause that specifies no routine name:

```
UPDATE STATISTICS FOR ROUTINE;
```

Running this statement reoptimizes the DML statement execution plans for every SPL routine in the database that operates on local tables.

Similarly, you can substitute the keyword FUNCTION for ROUTINE in the previous example to reoptimize execution plans only for SPL routines that return at least one value, or you can substitute the keyword FUNCTION for PROCEDURE to reoptimize execution plans only for SPL routines that return no value. In these cases, the database server does not update the statistics in the system catalog tables.

You do not need to run UPDATE STATISTICS statements on C or Java™ UDRs.

Verify the integrity of the reverted data

Before you allow users to access the databases, check the integrity of the reverted data.

Follow the steps in [Verifying the integrity of the data on page 14](#).

Back up the database server after reversion

After you complete the reversion, use ON-Bar or **ontape** to make a level-0 backup of the database server to which you reverted.

For more information about making backups, see your *Informix® Backup and Restore Guide*.



Important: Do not overwrite the tapes that you used to back up your source database server.

Return the database server to online mode

To bring the old database server online, run the **onmode -m** command.

Then users can access the data.

Register DataBlade® modules

You must register any DataBlade® modules that your databases require. Built-in database extensions automatically revert to the version included in the database server to which you reverted.

If you are reverting to version 11.70.xC1 or later, you do not need to register built-in database extensions because they are registered automatically.

For information on registering DataBlade® modules, see the *Informix® DataBlade® Module Installation and Registration Guide*.

Migration of data between database servers

Migrating database servers to a new operating system

When you migrate to a new operating system, you must choose a tool for migrating your data, you might need to make some adjustments to your tables, and you must review environment-dependent configuration parameters and environment variables.

Related information

Paths for migration to the new version

Choosing a tool for moving data before migrating between operating systems

If you are migrating between different operating systems, you must choose a method for exporting and importing data. The tool that you choose for exporting and importing data depends on how much data you plan to move.

All these methods deliver similar performance and enable you to modify the schema of the database. The tools that you can use include:

- The **dbexport** and **dbimport** utilities, which you can use to move an entire database
- The UNLOAD and LOAD statements, which move selected columns or tables (The LOAD statement does not change the data format.)
- The **dbload** utility, which you can use to change the data format
- The **onunload** utility, which unloads data in page-sized chunks, and the **onload** utility, which moves data to an identical database server on a computer of the same type
- Enterprise Replication, which you can use to transfer data between Informix® on one operating system and Informix® on a second operating system.

For an overview of all of these data-migration tools, a comparison of tools, and information about which versions of the database server do not support all of the tools, see [Data-migration tools on page 2](#).

Related information

[The dbexport and dbimport utilities on page 65](#)

[The onunload and onload utilities on page 117](#)

[The dbload utility on page 82](#)

[The dbschema utility on page 98](#)

[The LOAD and UNLOAD statements on page 115](#)

Adjusting database tables for file-system variations

File system limitations vary between NFS and non-NFS file systems. You might need to break up large tables when you migrate to a new operating system.

For example, if you have a 3 GB table, but your operating system allows only 2 GB files, break up your table into separate files before you migrate. For more information, see your *Informix® Administrator's Guide*.

The Informix® storage space can reside on an NFS-mounted file system using regular operating-system files. For information about the NFS products you can use to NFS mount a storage space for the Informix® database server, check product compatibility information.

Moving data to a database server on a different operating system

You can move data between Informix® database servers on UNIX™ or Linux™ and Windows™.

About this task

To move data to a database server on a different operating system:

1. Save a copy of the current configuration files.
2. Use ON-Bar or **ontape** to make a final level-0 backup.
For more information, refer to your *Informix® Backup and Restore Guide*.
3. Choose one of the following sets of migration utilities to unload the databases:
 - **dbexport** and **dbimport**
 - UNLOAD, **dbschema**, and LOAD
 - UNLOAD, **dbschema**, and **dbload**
4. Bring the source database server offline.
5. Install and configure the target database server. If you are migrating to Windows™, also install the administration tools.
6. Bring the target database server online.
7. Use **dbimport**, LOAD, or **dbload**, or external tables to load the databases into the target database server, depending on which utility you used to export the databases.
8. Make an initial level-0 backup of the target database server.
9. Run UPDATE STATISTICS to update the information that the target database server uses to plan efficient queries.

Results

Adapting your programs for a different operating system

When you change to a different operating system, you must review and, if necessary, adjust your environment-dependent configuration parameters and environment variables.

Certain database server configuration parameters and environment variables are environment-dependent.

For details, see the information about configuration parameters in the *Informix® Administrator's Guide* and the *Informix® Administrator's Reference* and the information about environment variables in the *Informix® Administrator's Guide* and the *Informix® Guide to SQL: Reference*.

Ensuring the successful creation of system databases

The first time the database server is brought online, the **sysmaster**, **sysutils**, **sysuser**, and **sysadmin** databases are built.

After moving to a database server on a different operating system, check the message log to ensure that the **sysmaster** and **sysutils** databases have been created successfully before you allow users to access the database server.

After you ensure that client users can access data on the database server, the migration process is complete.

Next you might want to seek ways to obtain maximum performance. For details on topics related to performance, see your *Informix® Performance Guide*.

Data migration utilities

External tables

External tables are a fast and versatile method for moving data between database servers. External tables are the fastest method for loading data into a RAW table with no indexes.

You run the CREATE EXTERNAL TABLE statement to define an external table that is not part of your database to unload data from your database. You run INSERT ... SELECT statements to load data from the external table into your database.

You can unload and load data in the internal Informix® representation. All Informix® data types are supported. You can define a value to be interpreted as a NULL when you load or unload data from an external table. You can specify the date and currency format and replace missing values with column defaults.

You define a named pipe to copy data from one Informix® instance to another without writing the data to an intermediate file. You can monitor the I/O queues to determine whether you have enough FIFO virtual processors. If necessary, you can add more FIFO virtual processors to improve performance. You can specify to run high-speed transfers in parallel.

Rows that have conversion errors during a load are written to a reject file on the server that performs the conversion.

The dbexport and dbimport utilities

The dbexport and dbimport utilities import and export a database and its schema to disk or tape.

The dbexport utility unloads an entire database into text files and creates a schema file. You can unload the database and its schema file either to disk or tape. If you prefer, you can unload the schema file to disk and unload the data to tape. You can use the schema file with the dbimport utility to re-create the database schema in another HCL Informix® environment, and you can edit the schema file to modify the database that dbimport creates.

The dbimport utility creates a database and loads it with data from text files on tape or disk. The input files consist of a schema file that is used to re-create the database and data files that contain the database data. Normally, you generate the input files with the dbexport utility, but you can use any properly formatted input files.



Attention:

When you import a database, use the same environment variable settings and configuration parameter settings that were used when the database was created.

- If any environment variables or configuration parameters that affect fragmentation, constraints, triggers, or user-defined routines are set differently than they were when these database objects were created originally, the database that is created by the dbimport utility might not be an accurate reproduction of the original.
- Incompatible settings are likely to occur if you move data from an earlier version of the database server to a newer version. Over time, some configuration parameters or environment variables are deprecated, or their default values are changed. For example, assume that attached indexes were created by default in the original database. In the current version of the database server, detached indexes are created by default. If you want to maintain the original behavior, you can set the **DEFAULT_ATTACH** environment variable to 1 before you run the dbimport utility.



Also, the `dbimport` operation might fail when you attempt to import from a higher server version to a lower server version if the database schema changed between versions. For example, the `am_expr_pushdown` column was added to the `sysams` system catalog table in Informix® 11.70. The `dbimport` operation will fail if you attempt to import a database from Informix® 12.10 that contains the `am_expr_pushdown` column into a database from Informix® 11.50 that is missing that column. In that case, you must review the messages in the `dbimport.out` file, which is in your current directory. After you address the issues that caused the `dbimport` operation to fail, run the `dbimport` command again.

Requirements or limitations apply in the following cases:

Compressed data

The `dbexport` utility uncompresses compressed data. You must recompress the data after you use the `dbimport` utility to import the data.

Date values

Use four-digit years for date values. The date context for an object includes the date that the object was created, the values of the `DBCENTURY` and `GL_DATE` environment variables, and some other environment variables. If the date context during import is not the same as when these objects were created, you might get explicit errors, you might not be able to find your data, or a check constraint might not work as expected. Some of these problems do not generate errors.



Tip: By default, the `dbexport` utility exports dates in four-digit years unless environment variables are set to values that would override that format.

High-availability clusters

You cannot use the `dbexport` utility on HDR secondary servers or shared disk (SD) secondary servers.

The `dbexport` utility is supported on a remote standalone (RS) secondary server only if the server is set to stop applying log files. Use the `STOP_APPLY` configuration parameter to stop application of log files.

The `dbimport` utility is supported on all updatable secondary servers.

Label-based access control (LBAC)

When you export data that is LBAC-protected, the data that is exported is limited to the data that your LBAC credentials allow you to read. If your LBAC credentials do not allow you to read a row, that row is not exported, but no error is returned. To export all the rows, you must be able to see all the rows.

NLSCASE mode

Whether the NLSCASE mode of your source database is `SENSITIVE` or `INSENSITIVE`, you can reduce the risk of case-sensitivity issues by always migrating to a target database that has the same NLSCASE mode as the source database. For tables that include columns with case-variant `NCHAR` and `NVARCHAR` data values (for example, 'IBM', 'ibm', 'Ibm'), you might encounter the following differences after migration:

- ORDER BY and sorting operations can produce a different ordering of qualifying rows in query results, compared to the result of the same query before migration.
- Unique indexes and referential constraints with which the data were compliant before the migration might have integrity violations in the new database, if any index or constraint key column contains case-variant forms of the same character string.
- Queries with predicates that apply conditional operators to NCHAR or NVARCHAR values might return different results from the same data after migration.

Nondefault database locales

If the database uses a nondefault locale and the **GL_DATETIME** environment variable has a nondefault setting, you must set the **USE_DTENV** environment variable to the value of **1** so that localized DATETIME values are processed correctly by the dbexport and dbimport utilities.

SELECT triggers on tables

You must disable SELECT triggers before you export a database with the dbexport utility. The dbexport utility runs SELECT statements during export. The SELECT statement triggers might modify the database content.

Virtual tables for the Informix® MQ extension

The MQCreateVtiRead(), MQCreateVtiReceive(), and MCQCreateVtiWrite() functions create virtual tables, and map them to the appropriate WebSphere® MQ message queue. When the dbexport utility unloads data, it removes the messages from WebSphere® MQ queues. Before you use the dbexport utility, drop any MQ virtual tables. After you load the database with the dbimport utility, you can create the tables in the target database by using the appropriate functions.

Related information

Paths for migration to the new version

[Data-migration tools on page 2](#)

[Choosing a tool for moving data before migrating between operating systems on page 62](#)

Syntax of the dbexport command

The dbexport command unloads a database into text files that you can later import into another database. The command also creates a schema file.

dbexport

-c -d

-no-data-tables *tablename*s

-no-data-tables-accessmethods *accessmethods*

-nw-q


Destination Options¹

-ss-si-X


database

-V

-version

Element	Purpose	Key Considerations
-c	Makes dbexport complete exporting unless a fatal error occurs	References: For details on this option, see dbexport errors on page 70 .
-d	Makes dbexport export simple-large-object descriptors only, not simple-large-object data	
-q	Suppresses the display of error messages, warnings, and generated SQL data-definition statements	None.
-ss	Generates database server-specific information for all tables in the specified database	References: For details on this option, see dbexport server-specific information on page 70 .
-si	Excludes the generation of index storage clauses for non-fragmented tables The -si option is available only with the -ss option.	References: For details on this option, see dbexport server-specific information on page 70 .
-X	Recognizes HEX binary data in character fields	None.
-no-data-tables	Prevents data from being exported for the specified tables. Only the definitions of the specified tables are exported.	Accepts a comma-separated list of names of tables for which data will not be exported.  Default behavior: Only the definition of the tsinstanceTable table is exported, not the data.

1. See [dbexport destination options on page 71](#)

Element	Purpose	Key Considerations
		 The data and definitions of all other tables are exported.
-no-data-tables-accessmethods	Prevents data from being unloaded using the specified access methods.	Accepts a comma-separated list of names of access methods. Tables using those access methods are not unloaded. Default value: ts_rts_vtam, ts_vtam Tables using ts_rts_vtam and ts_vtam access methods are not unloaded.
-nw	Generates the SQL for creating a database without the specification of an owner	None.
-V	Displays the software version number and the serial number	None.
-version	Extends the -V option to display additional information about the build operating system, build number, and build date	None.
database	Specifies the name of the database that you want to export	Additional information: If your locale is set to use multibyte characters, you can use multibyte characters for the database name. References: If you want to use more than the simple name of the database, see Database Name on page 80 .

You must have DBA privileges or log in as user **informix** to export a database.




Global Language Support: When the environment variables are set correctly, as described in the *Informix® GLS User's Guide*, dbexport can handle foreign characters in data and export the data from GLS databases. For more information, refer to [Database renaming on page 80](#).

You can set the IFX_UNLOAD_EILSEQ_MODE environment variable to enable dbexport to use character data that is invalid for the locale specified in the environment.

You can use delimited identifiers with the dbexport utility. The utility detects database objects that are keywords, mixed case, or have special characters, and the utility places double quotes around them.

In addition to the data files and the schema file, dbexport creates a file of messages named `dbexport.out` in the current directory. This file contains error messages, warnings, and a display of the SQL data definition statements that it generates. The same material is also written to standard output unless you specify the **-q** option.

During export, the database is locked in exclusive mode. If `dbexport` cannot obtain an exclusive lock, it displays a diagnostic message and exits.

 **Tip:** The `dbexport` utility can create files larger than 2 GB. To support such large files, make sure your operating system file-size limits are set sufficiently high. For example, on UNIX™, set `ulimit` to unlimited.

Example

Example

The following command exports the table definitions but no data for all the tables in the `customer` database.

```
dbexport -no-data-tables -no-data-tables-accessmethods customer
```

Example

Example

The following command generates the schema and data for the `customer` database without the specification of an owner:

```
dbexport customer -nw
```

Termination of the `dbexport` utility

You can stop the `dbexport` utility at any time.

To cancel `dbexport`, press your `Interrupt` key.

The `dbexport` utility asks for confirmation before it terminates.

`dbexport` errors

The `dbexport -c` option tells `dbexport` to complete exporting unless a fatal error occurs.

Even if you use the `-c` option, `dbexport` interrupts processing if one of the following fatal errors occurs:

- `dbexport` is unable to open the specified tape.
- `dbexport` finds bad writes to the tape or disk.
- Invalid command parameters were used.
- `dbexport` cannot open the database or there is no system permission for doing so.
- A subdirectory with the name specified during invocation already exists

dbexport server-specific information

The **dbexport -ss** option generates server-specific information. This option specifies initial- and next-extent sizes, fragmentation information if the table is fragmented, the locking mode, the dbspace for a table, the blob space for any simple large objects, and the dbspace for any smart large objects.

The **dbexport -si** option, which is available only with the **-ss** option, does not generate index storage clauses for non-fragmented tables.

dbexport destination options

The dbexport utility supports disk and tape destination options.

Destination options

-odirectory

-tdevice -bblocksize -stapesize

-fpathname

Element	Purpose	Key Considerations
-b <i>blocksize</i>	Specifies, in kilobytes, the block size of the tape device.	None.
-f <i>pathname</i>	Specifies the name of the path where you want the schema file stored, if you are storing the data files on tape.	The path name can be a complete path name or a file name. If only a file name is given, the file is stored in the current directory. If you do not use the -f option, the SQL source code is written to the tape.
-o <i>directory</i>	Specifies the directory on disk in which dbexport creates the <i>database.exp</i> directory. This directory holds the data files and the schema file that dbexport creates for the database.	The specified directory must exist.
-s <i>tapesize</i>	Specifies, in kilobytes, the amount of data that you can store on the tape.	To write to the end of the tape, set the value to 0. If you do not specify 0, the maximum size is 2 097 151 KB.
-t <i>device</i>	Specifies the path name of the tape device where you want the text files and, possibly, the schema file stored.	You cannot specify a remote tape device.

When you write to disk, dbexport creates a subdirectory, *database.exp*, in the directory that the **-o** option specifies. The dbexport utility creates a file with the *.unl* extension for each table in the database. The schema file is written to the file *database.sql*. The *.unl* and *.sql* files are in the *database.exp* directory.

If you do not specify a destination for the data and schema files, the subdirectory `database.exp` is placed in the current working directory.

When you write the data files to tape, you can use the `-f` option to store the schema file to disk. You are not required to name the schema file `database.sql`. You can give it any name.

UNIX/Linux Only

For database servers on UNIX™ or Linux™, the command is:

```
dbexport //finland/reports
```

The following command exports the database **stores_demo** to tape with a block size of 16 KB and a tape capacity of 24 000 KB. The command also writes the schema file to `/tmp/stores_demo.imp`.

```
dbexport -t /dev/rmt0 -b 16 -s 24000 -f /tmp/stores_demo.imp  
stores_demo
```

The following command exports the same **stores_demo** database to the directory named `/work/exports/stores_demo.exp`. The resulting schema file is `/work/exports/stores_demo.exp/stores_demo.sql`.

```
dbexport -o /work/exports stores_demo
```

Windows™ Only

For Windows™, the following command exports the database **stores_demo** to tape with a block size of 16 KB and a tape capacity of 24 000 KB. The schema file is written to `C:\temp\stores_demo.imp`.

```
dbexport -t \\.\TAPE2 -b 16 -s 24000 -f  
C:\temp\stores_demo.imp stores_demo
```

The following command exports the same **stores_demo** database to the directory named `D:\work\exports\stores_demo.exp`. The resulting schema file is `D:\work\exports\stores_demo.exp\stores_demo.sql`.

```
dbexport -o D:\work\exports stores_demo
```

Exporting time series data in rolling window containers

The `dbexport` utility exports time series data except any data that is in the dormant window of rolling window containers.

About this task

The active window in rolling window containers is re-created after the time series data is loaded into a container.

The dormant window is not exported. To export the data from the dormant window, you must move the data into the active window.

To export time series data in the dormant window of a rolling window container:

1. If necessary, increase the size of the active window by running the TSContainerManage function. The size of the active window must be large enough to fit all the intervals in the dormant window can fit into the active window.
2. Move the intervals in the dormant window into the active window by running the TSContainerManage function.
3. Export the data by running the dbexport utility.

Related information

[Contents of the schema file that dbexport creates on page 73](#)

Contents of the schema file that dbexport creates

The dbexport utility creates a schema file. This file contains the SQL statements that you need to re-create the exported database.

You can edit the schema file to modify the schema of the database.

If you use the -ss option, the schema file contains server-specific information, such as initial- and next-extent sizes, fragmentation information, lock mode, the dbspace where each table resides, the blob space where each simple-large-object column resides, and the dbspace for smart large objects. The following information is not retained:

- Logging mode of the database
 - For information about logging modes, see the *Informix® Guide to SQL: Reference*.
- The starting values of SERIAL columns
- The dormant window interval values for time series rolling window containers

The statements in the schema file that create tables, views, indexes, partition-fragmented tables and indexes, roles, and grant privileges do so with the name of the user who originally created the database. In this way, the original owner retains DBA privileges for the database and is the owner of all the tables, indexes, and views. In addition, the person who runs the dbimport command also has DBA privileges for the database.

The schema file that dbexport creates contains comments, which are enclosed in braces, with information about the number of rows, columns, and indexes in tables, and information about the unload files. The dbimport utility uses the information in these comments to load the database.

The number of rows must match in the unload file and the corresponding unload comment in the schema file. If you change the number of rows in the unload file but not the number of rows in the schema file, a mismatch occurs.



Attention: Do not delete any comments in the schema file, and do not change any existing comments or add any new comments. If you change or add comments, the dbimport utility might stop or produce unpredictable results.

If you delete rows from an unload file, update the comment in the schema file with the correct number of rows in the unload file. Then dbimport is successful.

Related information

[Exporting time series data in rolling window containers on page 72](#)

Syntax of the dbimport command

The dbimport command imports previously exported data into another database.

dbimport

-c -D -nv -q -X

Input-File Location ²

Create Options ³

-V

-version

database

Element	Purpose	Key Considerations
-c	Completes importing data even when certain nonfatal errors occur	For more information, see dbimport errors and warnings on page 76 .
-D	Specifies a default extent size of 16 KB for the first and subsequent extents during the import operation, if the extent sizes are not specified in the CREATE TABLE statement.	<p>This option is ignored if the extent sizes are specified in the CREATE TABLE statement.</p> <p>Default values help to ensure that enough space is available in the dbspace that is designated for the import operation.</p> <p>This option prevents the automatic calculation of extent sizes during the import operation, and is useful especially in the following situations:</p> <ul style="list-style-type: none"> • When importing tables that contain columns with large maximum row sizes, such as LVARCHAR columns. • When importing data after the dbexport command was run without the -ss option. The -ss option specifies server-specific information about extent sizes.
-nv	While the dbimport -nv command is running, tables with foreign-key constraints that ALTER TABLE ADD CONSTRAINT creates in enabled or filtering mode are not checked	By bypassing the checking of referential constraints, this option can reduce migration time for very large tables that already conform to their foreign-key constraints. The

2. See [dbimport input-file location options on page 76](#)

3. See [dbimport create options on page 78](#)

Element	Purpose	Key Considerations
	for violations, as if you had also specified NOVALIDATE	NOVALIDATE mode does not persist after the ALTER TABLE ADD CONSTRAINT statement has completed.
-q	Suppresses the display of error messages, warnings, and generated SQL data-definition statements	None.
-V	Displays the software version number and the serial number	None.
-version	Extends the -V option to display additional information about the build operating system, build number, and build date	None.
-X	Recognizes HEX binary data in character fields	None.
<i>database</i>	Declares the name of the database to create	If you want to use more than the simple name of the database, see Database Name on page .

The dbimport utility can use files from the following location options:

- All input files are on disk.
- All input files are on tape.
- The schema file is on disk, and the data files are on tape.



Important: Do not put comments into your input file. Comments might cause unpredictable results when the dbimport utility reads them.

The dbimport utility supports the following tasks for an imported Informix® database server:

- Specify the dbspace where the database will reside
- Create an ANSI-compliant database with unbuffered logging
- Create a database that supports explicit transactions (with buffered or unbuffered logging)
- Create an unlogged database
- Create a database with the NLS case-insensitive property for NCHAR and NVARCHAR strings.
- Process all ALTER TABLE ADD CONSTRAINT and SET CONSTRAINTS statements in the .sql file of the exported database that define enabled or filtering referential constraints so that any foreign-key constraints that are not specified as DISABLED are in ENABLED NOVALIDATE or in FILTERING NOVALIDATE mode.



Note: If you specify the -nv option, the .sql file of the exported database is not modified, but any foreign-key constraints that ALTER TABLE ADD CONSTRAINT or SET CONSTRAINTS statements enable are processed without checking each row of the table for violations. The `ENABLED, OF FILTERING WITH ERROR, OF FILTERING WITHOUT ERROR` constraint mode specifications are implemented instead as the `ENABLED NOVALIDATE, OF FILTERING WITH ERROR`



`NOVALIDATE` or `FILTERING WITHOUT ERROR NOVALIDATE` modes. After the foreign-key constraints have been enabled without checking for violations, their modes automatically revert to whatever the `.sql` file specified so that subsequent DML operations on the tables enforce referential integrity.

The user who runs the `dbimport` utility is granted the DBA privilege on the newly created database. The `dbimport` process locks each table as it is being loaded and unlocks the table when the loading is complete.



Global Language Support: When the GLS environment variables are set correctly, as the *Informix® GLS User's Guide* describes, `dbimport` can import data into database server versions that support GLS.

Termination of the `dbimport` utility

You can stop the `dbimport` utility at any time.

To cancel the `dbimport` utility, press your `Interrupt` key.

The `dbimport` utility asks for confirmation before it terminates.

`dbimport` errors and warnings

The `dbimport -c` option tells the `dbimport` utility to complete exporting unless a fatal error occurs.

If you include the `-c` option in a `dbimport` command, `dbimport` ignores the following errors:

- A data row that contains too many columns
- Inability to put a lock on a table
- Inability to release a lock

Even if you use the `-c` option, `dbimport` interrupts processing if one of the following fatal errors occurs:

- Unable to open the tape device specified
- Bad writes to the tape or disk
- Invalid command parameters
- Cannot open database or no system permission
- Cannot convert the data

The `dbimport` utility creates a file of messages called `dbimport.out` in the current directory. This file contains any error messages and warnings that are related to `dbimport` processing. The same information is also written to the standard output unless you specify the `-q` option.

dbimport input-file location options

The input-file location specifies the location of the `database.exp` directory, which contains the files that the dbimport utility imports.

If you do not specify an input-file location, dbimport searches for data files in the directory `database.exp` under the current directory and for the schema file in `database.exp/database.sql`.

dbimport input-file location

`-idirectory`

`-tdevice`

`-bblocksize-s tapesize`

`-fpathname`

Element	Purpose	Key Considerations
<code>-b blocksize</code>	Specifies, in kilobytes, the block size of the tape device	If you are importing from tape, you must use the same block size that you used to export the database. If you do not use the <code>-b</code> option, the default block size is 1.
<code>-f pathname</code>	Specifies where dbimport can find the schema file to use as input to create the database when the data files are read from tape	If you use the <code>-f</code> option to export a database, you typically use the same path name that you specified in the dbexport command. If you specify only a file name, dbimport looks for the file in the <code>.exp</code> subdirectory of your current directory. If you do not use the <code>-f</code> option, the SQL source code is written to the tape.
<code>-i directory</code>	Specifies the complete path name on disk of the <code>database.exp</code> directory, which holds the input data files and schema file that dbimport uses to create and load the new database. The directory name must be the same as the database name.	This directory must be the same directory that you specified with the dbexport <code>-o</code> option. If you change the directory name, you also rename your database.
<code>-s tapesize</code>	Specifies, in kilobytes, the amount of data that you can store on the tape	To read to the end of the tape, specify a tape size of 0. If you are importing from tape, you must use the same tape size that you used to export the database. The maximum size is 2 097 151 KB. If you do not use the <code>-s</code> option, the default value is 0 (read to the end of the tape).
<code>-t device</code>	Specifies the path name of the tape device that holds the input files	You cannot specify a remote tape device.

Examples showing input file location on UNIX™ or Linux™

To import the **stores_demo** database from a tape with a block size of 16 KB and a capacity of 24 000 KB, issue this command:

```
dbimport -c -t /dev/rmt0 -b 16 -s 24000 -f
/tmp/stores_demo.imp stores_demo
```

The schema file is read from `/tmp/stores_demo.imp`.

To import the **stores_demo** database from the `stores_demo.exp` directory under the `/work/exports` directory, issue this command:

```
dbimport -c -i /work/exports stores_demo
```

The schema file is assumed to be `/work/exports/stores_demo.exp/stores_demo.sql`.

Examples showing input file location on Windows™

To import the **stores_demo** database from a tape with a block size of 16 KB and a capacity of 24 000 KB, issue this command:

```
dbimport -c -t \\.\TAPEDRIVE -b 16 -s 24000 -f
C:\temp\stores_demo.imp stores_demo
```

The schema file is read from `C:\temp\stores_demo.imp`.

To import the **stores_demo** database from the `stores_demo.exp` directory under the `D:\work\exports` directory, issue this command:

```
dbimport -c -i D:\work\exports stores_demo
```

The schema file is assumed to be `D:\work\exports\stores_demo.exp\stores_demo.sql`.

dbimport create options

The **dbimport** utility supports options for creating a database, specifying a dbspace for that database, defining logging options, and optionally specifying ANSI/ISO-compliance or NLS case-insensitivity (or both) as properties of the database.

Create options

-dbspace

-l

buffered

-ansi

-ci

Element	Purpose	Key Considerations
-ansi	Creates an ANSI/ISO-compliant database in which the ANSI/ISO rules for transaction logging are enabled. Otherwise, the	If you omit the -ansi option, the database uses explicit transactions.

Element	Purpose	Key Considerations
	database uses explicit transactions by default.	Additional Information: For more information about ANSI/ISO-compliant databases, see the <i>Informix® Guide to SQL: Reference</i> .
-ci	Specifies the NLS case-insensitive property. Otherwise, the database is case-sensitive by default.	Additional Information: See the <i>Informix® Guide to SQL: Syntax</i> and <i>Informix® Guide to SQL: Reference</i> descriptions of the NLS case-insensitive property.
-d dbspace	Specifies the dbspace where the database is created. .	If this is omitted, the default location is the root dbspace
-l	Establishes unbuffered transaction logging for the imported database. If the -l flag is omitted, the database is unlogged,	References: For more information, see Database-logging mode on page 80 .
-l buffered	Establishes buffered transaction logging for the imported database. If -l is included but buffered is omitted, the database uses unbuffered logging.	References: For more information, see Database-logging mode on page 80 .

If you created a table or index fragment containing partitions in Informix® Version 10.00 or a later version of the Informix® database server, you must use syntax containing the partition name when importing a database that contains multiple partitions within a single dbspace. See the *Informix® Guide to SQL: Syntax* for syntax details.

Example showing dbimport create options (UNIX™ or Linux™)

To import the **stores_demo** database from the **/usr/informix/port/stores_demo.exp** directory, issue this command:

```
dbimport -c stores_demo -i /usr/informix/port -l -ansi
```

The new database is ANSI/ISO-compliant.

The next example similarly imports the **stores_demo** database from the **/usr/informix/port/stores_demo.exp** directory. The imported database uses buffered transaction logging and explicit transactions. The **-ci** flag specifies *case insensitivity* in queries and in other operations on columns and character strings of the NCHAR and NVARCHAR data types:

```
dbimport -c stores_demo -i /usr/informix/port -l buffered -ci
```

The **-ansi** and **-ci** options for database properties are not mutually exclusive. You can specify an ANSI/ISO-compliant database that is also NLS case-insensitive, as in the following example of the **dbimport** command:

```
dbimport -c stores_demo -i /usr/informix/port -l -ansi -ci
```

Example showing dbimport create options (Windows™)

To import the **stores_demo** database from the **C:\USER\informix\port\stores_demo.exp** directory, issue this command:

```
dbimport -c stores_demo -i C:\USER\informix\port -l -ansi
```

The imported database is ANSI/ISO-compliant and is case-sensitive for all built-in character data types.

Database-logging mode

Because the logging mode is not retained in the schema file, you can specify logging information when you use the **dbimport** utility to import a database.

You can specify any of the following logging options when you use **dbimport**:

- ANSI-compliant database with unbuffered logging
- Unbuffered logging
- Buffered logging
- No logging

For more information, see [dbimport create options on page 78](#).

The **-l** options are equivalent to the logging clauses of the CREATE DATABASE statement, as follows:

- Omitting any of the **-l** options is equivalent to omitting the WITH LOG clause.
- The **-l** option is equivalent to the WITH LOG clause.
- The **-l buffered** option is equivalent to the WITH BUFFERED LOG.
- The **-l ansi** option is equivalent to the WITH LOG MODE ANSI clause, and implies unbuffered logging.

Database renaming

The **dbimport** utility gives the new database the same name as the database that you exported. If you export a database to tape, you cannot change its name when you import it with **dbimport**. If you export a database to disk, you can change the database name.

You can use the RENAME DATABASE statement to change the database name.

Alternative ways to change the database name

The following examples show alternative ways to change the database name. In this example, assume that **dbexport** unloaded the database **stores_demo** into the directory **/work/exports/stores_demo.exp**. Thus, the data files (the **.unl** files) are stored in **/work/exports/stores_demo.exp**, and the schema file is **/work/exports/stores_demo.exp/stores_demo.sql**.

To change the database name to a new name on UNIX™ or Linux™:

1. Change the name of the **.exp** directory. That is, change **/work/exports/stores_demo.exp** to **/work/exports/newname.exp**.
2. Change the name of the schema file. That is, change **/work/exports/stores_demo.exp/stores_demo.sql** to **/work/exports/stores_demo.exp/newname.sql**. Do not change the names of the **.unl** files.
3. Import the database with the following command:

```
dbimport -i /work/exports newname
```

To change the database name to a new name on Windows™:

In the following example, assume that **dbexport** unloaded the database **stores_demo** into the directory **D:\work\exports\stores_demo.exp**. Thus, the data files (the **.unl** files) are stored in **D:\work\exports\stores_demo.exp**, and the schema file is **D:\work\exports\stores_demo.exp\stores_demo.sql**.

1. Change the name of the **.exp** directory. That is, change **D:\work\exports\stores_demo.exp** to **D:\work\exports\newname.exp**.
2. Change the name of the schema file. That is, change **D:\work\exports\stores_demo.exp\stores_demo.sql** to **D:\work\exports\stores_demo.exp\newname.sql**. Do not change the names of the **.unl** files.
3. Import the database with the following command:

```
dbimport -i D:\work\exports
```

Changing the database locale with dbimport

You can use the **dbimport** utility to change the locale of a database.

To change the locale of a database:

1. Set the **DB_LOCALE** environment variable to the name of the current database locale.
2. Run **dbexport** on the database.
3. Use the DROP DATABASE statement to drop the database that has the current locale name.
4. Set the **DB_LOCALE** environment variable to the desired database locale for the database.
5. Run **dbimport** to create a new database with the desired locale and import the data into this database.

Simple large objects

When the **dbimport**, **dbexport**, and DB-Access utilities process simple-large-object data, they create temporary files for that data in a temporary directory.

Before you export or import data from tables that contain simple large objects, you must have one of the following items:

- A **\tmp** directory on your currently active drive
- The **DBTEMP** environment variable set to point to a directory that is available for temporary storage of the simple large objects

Windows™ Only

Windows™ sets the **TMP** and **TEMP** environment variables in the command prompt sessions, by default.

However, if the **TMP**, **TEMP**, and **DBTEMP** environment variables are not set, **dbimport** places the temporary files for the simple large objects in the **\tmp** directory.



Attention: If a table has a CLOB or BLOB in a column, you cannot use **dbexport** to export the table to a tape. If a table has a user-defined type in a column, using **dbexport** to export the table to a tape might yield unpredictable results, depending on the export function of the user-defined type. Exported CLOB sizes are stored in hex format in the unload file.

The dbload utility

The **dbload** utility loads data into databases or tables that Informix® products created. It transfers data from one or more text files into one or more existing tables.

Prerequisites: If the database contains label-based access control (LBAC) objects, the **dbload** utility can load only those rows in which your security label dominates the column-security label or the row-security label. If the entire table is to be loaded, you must have the necessary LBAC credentials for writing all of the labeled rows and columns. For more information about LBAC objects, see the *Informix® Security Guide* and the *Informix® Guide to SQL: Syntax*.

You cannot use the **dbload** utility on secondary servers in high-availability clusters.

When you use the **dbload** utility, you can manipulate a data file that you are loading or access a database while it is loading. When possible, use the LOAD statement, which is faster than **dbload**.

The **dbload** utility gives you a great deal of flexibility, but it is not as fast as the other methods, and you must prepare a command file to control the input. You can use **dbload** with data in a variety of formats.

The **dbload** utility offers the following advantages over the LOAD statement:

- You can use **dbload** to load data from input files that were created with a variety of format arrangements. The **dbload** command file can accommodate data from entirely different database management systems.
- You can specify a starting point in the load by directing **dbload** to read but ignore x number of rows.
- You can specify a batch size so that after every x number of rows are inserted, the insert is committed.
- You can limit the number of bad rows read, beyond which **dbload** ends.

The cost of **dbload** flexibility is the time and effort spent creating the **dbload** command file, which is required for **dbload** operation. The input files are not specified as part of the **dbload** command line, and neither are the tables into which the data is inserted. This information is contained in the command file.

Related information

[Data-migration tools on page 2](#)

[Choosing a tool for moving data before migrating between operating systems on page 62](#)

Syntax of the dbload command

The **dbload** command loads data into databases or tables.

dbload**-d***database***-c***command file***-l***error log file***-r-k****-e***errors-p***-i***ignore rows***-n***commit interval***-X****-s****-V****-version**

Element	Purpose	Key Considerations
-c <i>command file</i>	Specifies the file name or path name of a dbload command file	References: For information about building the command file, see Command file for the dbload utility on page 86 .
-d <i>database</i>	Specifies the name of the database to receive the data	Additional Information: To use more than the simple name of the database, see Database Name on page 86 .
-e <i>errors</i>	Specifies the number of bad rows that dbload reads before terminating. The default value for <i>errors</i> is 10.	References: For more information, see Bad-row limit during a load operation on page 85 .
-i <i>ignore rows</i>	Specifies the number of rows to ignore in the input file	References: For more information, see Rows to ignore during a load operation on page 85 .
-k	Instructs dbload to lock the tables listed in the command file in exclusive mode during the load operation	References: For more information, see Table locking during a load operation on page 84 . You cannot use the -k option with the -r option because the -r option specifies that no tables are locked during the load operation.
-l <i>error log file</i>	Specifies the file name or path name of an error log file	If you specify an existing file, its contents are overwritten. If you specify a file that does not exist, dbload creates the file. Additional Information: The error log file stores diagnostic information and any input file rows that dbload cannot insert into the database.
-n <i>commit interval</i>	Specifies the commit interval in number of rows The default interval is 100 rows.	Additional Information: If your database supports transactions, dbload commits a transaction after the specified number of new rows is read and inserted. A message appears after each commit.

Element	Purpose	Key Considerations
		References: For information about transactions, see the <i>Informix® Guide to SQL: Tutorial</i> .
-p	Prompts for instructions if the number of bad rows exceeds the limit	References: For more information, see Bad-row limit during a load operation on page 85 .
-r	Prevents dbload from locking the tables during a load, thus enabling other users to update data in the table during the load	Additional Information: For more information, see Table locking during a load operation on page 84 . You cannot use the -r option with the -k option because the -r option specifies that the tables are not locked during the load operation while the -k option specifies that the tables are locked in exclusive mode.
-s	Checks the syntax of the statements in the command file without inserting data	Additional Information: The standard output displays the command file with any errors marked where they are found.
-V	Displays the software version number and the serial number	None.
-version	Extends the -V option to display additional information about the build operating system, build number, and build date	None.
-X	Recognizes HEX binary data in character fields	None.



Tip: If you specify part (but not all) of the required information, **dbload** prompts you for additional specifications. The database name, command file, and error log file are all required. If you are missing all three options, you receive an error message.

Example

dbload command example

The following command loads data into the **stores_demo** database in the **turku** directory on a database server called **finland**:

```
dbload -d //finland/turku/stores_demo -c commands -l errlog
```

Table locking during a load operation

The **dbload -k** option overrides the default table lock mode during the load operation. The **-k** option instructs **dbload** to lock the tables in exclusive mode rather than shared mode.

If you do not specify the **-k** option, the tables specified in the command file are locked in shared mode. When tables are locked in shared mode, the database server still must acquire exclusive row or page locks when it inserts rows into the table.

When you specify the **-k** option, the database server places an exclusive lock on the entire table. The **-k** option increases performance for large loads because the database server does not need to acquire exclusive locks on rows or pages as it inserts rows during the load operation.

If you do not specify the **-r** option, the tables specified in the command file are locked during loading so that other users cannot update data in the table. Table locking reduces the number of locks needed during the load but reduces concurrency. If you are planning to load a large number of rows, use table locking and load during nonpeak hours.

Rows to ignore during a load operation

The **dbload -i** option specifies the number of new-line characters in the input file that **dbload** ignores before **dbload** begins to process data.

This option is useful if your most recent **dbload** session ended prematurely.

For example, if **dbload** ends after it inserts 240 lines of input, you can begin to load again at line 241 if you set *number rows ignore* to 240.

The **-i** option is also useful if header information in the input file precedes the data records.

Bad-row limit during a load operation

The **dbload -e** option lets you specify how many bad rows to allow before **dbload** terminates.

If you set *errors* to a positive integer, **dbload** terminates when it reads (*errors* + 1) bad rows. If you set *errors* to zero, **dbload** terminates when it reads the first bad row.

If **dbload** exceeds the bad-row limit and the **-p** option is specified, **dbload** prompts you for instructions before it terminates. The prompt asks whether you want to roll back or to commit all rows that were inserted since the last transaction.

If **dbload** exceeds the bad-row limit and the **-p** option is not specified, **dbload** commits all rows that were inserted since the last transaction.

Termination of the dbload utility

If you press your `Interrupt` key, **dbload** terminates and discards any new rows that were inserted but not yet committed to the database (if the database has transactions).

Name and object guidelines for the dbload utility

You must follow guidelines for specifying network names and handling simple large objects, indexes, and delimited identifiers when you use the **dbload** utility.

Table 8. Name and object guidelines for the dbload utility

Objects	Guideline
Network names	If you are on a network, include the database server name and directory path with the database name to specify a database on another database server.
Simple large objects	You can load simple large objects with the dbload utility as long as the simple large objects are in text files.
Indexes	The presence of indexes greatly affects the speed with which the dbload utility loads data. For best performance, drop any indexes on the tables that receive the data before you run dbload . You can create new indexes after dbload has finished.
Delimited identifiers	You can use delimited identifiers with the dbload utility. The utility detects database objects that are keywords, mixed case, or have special characters, and places double quotes around them. If your most recent dbload session ended prematurely, specify the starting line number in the command-line syntax to resume loading with the next record in the file.

Command file for the dbload utility

Before you use the **dbload** utility, you must create a command file that names the input data files and the tables that receive the data. The command file maps fields from one or more input files into columns of one or more tables within your database.

The command file contains only FILE and INSERT statements. Each FILE statement names an input data file. The FILE statement also defines the data fields from the input file that are inserted into the table. Each INSERT statement names a table to receive the data. The INSERT statement also defines how **dbload** places the data that is described in the FILE statement into the table columns.

Within the command file, the FILE statement can appear in these forms:

- Delimiter form
- Character-position form

The FILE statement has a size limit of 4,096 bytes.

Use the delimiter form of the FILE statement when every field in the input data row uses the same delimiter and every row ends with a new-line character. This format is typical of data rows with variable-length fields. You can also use the delimiter form of the FILE statement with fixed-length fields as long as the data rows meet the delimiter and new line requirements. The delimiter form of the FILE and INSERT statements is easier to use than the character-position form.

Use the character-position form of the FILE statement when you cannot rely on delimiters and you must identify the input data fields by character position within the input row. For example, use this form to indicate that the first input data field begins at character position 1 and continues until character position 20. You can also use this form if you must translate a character string into a null value. For example, if your input data file uses a sequence of blanks to indicate a null value, you must use this form if you want to instruct **dbload** to substitute null at every occurrence of the blank-character string.

You can use both forms of the FILE statement in a single command file. For clarity, however, the two forms are described separately in sections that follow.

Delimiter form of the FILE and INSERT statements

The FILE and INSERT statements that define information for the **dbload** utility can appear in a delimiter form.

The following example of a **dbload** command file illustrates a simple delimiter form of the FILE and INSERT statements. The example is based on the **stores_demo** database. An UNLOAD statement created the three input data files, **stock.unl**, **customer.unl**, and **manufact.unl**.

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO stock;
FILE customer.unl DELIMITER '|' 10;
INSERT INTO customer;
FILE manufact.unl DELIMITER '|' 3;
INSERT INTO manufact;
```

To see the **.unl** input data files, refer to the directory **\$INFORMIXDIR/demo/prod_name** (UNIX™ or Linux™) or **%INFORMIXDIR%\demo\prod_name** (Windows™).

Syntax for the delimiter form

The syntax for the delimiter form specifies the field delimiter, the input file, and the number of fields in each row of data.

The following diagram shows the syntax of the delimiter FILE statement.

FILE*filename***DELIMITER**

'*c*'

nfields

Element	Purpose	Key Considerations
<i>c</i>	Specifies the character as the field delimiter for the specific input file	<p>If the delimiter specified by <i>c</i> appears as a literal character anywhere in the input file, the character must be preceded with a backslash (\) in the input file. For example, if the value of <i>c</i> is specified as a square bracket ([), you must place a backslash before any literal square bracket that appears in the input file. Similarly, you must precede any backslash that appears in the input file with an additional backslash.</p> <p>You can specify any printable character, as defined by current locale, the tab character <code>TAB</code> (CTRL-I), or a blank space (ASCII 32) as the delimiter</p>

Element	Purpose	Key Considerations
		symbol. You cannot specify non-printable character, a hexadecimal character, or a backslash character.
<i>filename</i>	Specifies the input file	None.
<i>nfields</i>	Indicates the number of fields in each data row	None.

The **dbload** utility assigns the sequential names **f01**, **f02**, **f03**, and so on to fields in the input file. You cannot see these names, but if you refer to these fields to specify a value list in an associated INSERT statement, you must use the **f01**, **f02**, **f03** format. For details, refer to [How to write a dbload command file in delimiter form on page 89](#).

Two consecutive delimiters define a null field. As a precaution, you can place a delimiter immediately before the new-line character that marks the end of each data row. If the last field of a data row has data, you must use a delimiter. If you omit this delimiter, an error results whenever the last field of a data row is not empty.

Inserted data types correspond to the explicit or default column list. If the data field width is different from its corresponding character column width, the data is made to fit. That is, inserted values are padded with blanks if the data is not wide enough for the column or truncated if the data is too wide for the column.

If the number of columns named is fewer than the number of columns in the table, **dbload** inserts the default value that was specified when the table was created for the unnamed columns. If no default value is specified, **dbload** attempts to insert a null value. If the attempt violates a not null restriction or a unique constraint, the insert fails, and an error message is returned.

If the INSERT statement omits the column names, the default INSERT specification is every column in the named table. If the INSERT statement omits the VALUES clause, the default INSERT specification is every field of the previous FILE statement.

An error results if the number of column names listed (or implied by default) does not match the number of values listed (or implied by default).

The syntax of **dbload** INSERT statements resembles INSERT statements in SQL, except that in **dbload**, INSERT statements cannot incorporate SELECT statements.

Do not use the CURRENT, TODAY, and USER keywords of the INSERT INTO statement in a **dbload** command file; they are not supported in the **dbload** command file. These keywords are supported in SQL only.

For example, the following **dbload** command is not supported:

```
FILE "testtbl2.unl" DELIMITER '|' 1;
  INSERT INTO testtbl
    (testuser, testtime, testfield)
  VALUES
    ('kae', CURRENT, f01);
```

Load the existing data first and then write an SQL query to insert or update the data with the current time, date, or user login. You could write the following SQL statement:

```
INSERT INTO testtbl
      (testuser, testtime, testfield)
VALUES
      ('kae', CURRENT, f01);
```

The **CURRENT** keyword returns the system date and time. The **TODAY** keyword returns the system date. The **USER** keyword returns the user login name.

The following diagram shows the syntax of the **dbload** INSERT statement for delimiter form.

INSERT INTO

owner.

table

(

,column

)

VALUES clause ⁴

;

Element	Purpose	Key Considerations
<i>column</i>	Specifies the column that receives the new data	None.
<i>owner.</i>	Specifies the user name of the table owner	None.
<i>table</i>	Specifies the table that receives the new data	None.

Users who run **dbload** with this command file must have the Insert privilege on the named table.

How to write a dbload command file in delimiter form

Command files must contain required elements, including delimiters.

The FILE statement in the following example describes the **stock.unl** data rows as composed of six fields, each separated by a vertical bar (|) as the delimiter.

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO stock;
```

Two consecutive delimiters define a null field. As a precaution, you can place a delimiter immediately before the new-line character that marks the end of each data row. If the last field of a data row has data, you must use a delimiter. If you omit this delimiter, an error results.

Compare the FILE statement with the data rows in the following example, which appear in the input file **stock.unl**. (Because the last field is not followed by a delimiter, an error results if any data row ends with an empty field.)

```
1|SMT|baseball gLoves|450.00|case|10 gLoves/case
2|HRO|baseball|126.00|case|24/case
3|SHK|baseball bat|240.00|case|12/case
```

4. See [VALUES Clause on page](#) .

The example INSERT statement contains only the required elements. Because the column list is omitted, the INSERT statement implies that values are to be inserted into every field in the **stock** table. Because the VALUES clause is omitted, the INSERT statement implies that the input values for every field are defined in the most recent FILE statement. This INSERT statement is valid because the **stock** table contains six fields, which correspond to the number of values that the FILE statement defines.


The following example shows the first data row that is inserted into **stock** from this INSERT statement.

Field	Column	Value
f01	stock_num	1
f02	manu_code	SMT
f03	description	baseball gloves
f04	unit_price	450.00
f05	unit	case
f06	unit_descr	10 gloves/case

The FILE and INSERT statement in the following example illustrates a more complex INSERT statement syntax:

```
FILE stock.unl DELIMITER '|' 6;
INSERT INTO new_stock (col1, col2, col3, col5, col6)
VALUES (f01, f03, f02, f05, 'autographed');
```

In this example, the VALUES clause uses the field names that **dbload** assigns automatically. You must reference the automatically assigned field names with the letter **f** followed by a number: **f01, f02, f10, f100, f999, f1000**, and so on. All other formats are incorrect.

 **Tip:** The first nine fields must include a zero: f01, f02, ..., f09.

The user changed the column names, the order of the data, and the meaning of **col6** in the new **stock** table. Because the fourth column in **new_stock** (**col4**) is not named in the column list, the new data row contains a null value in the **col4** position (assuming that the column permits null values). If no default is specified for **col4**, the inserted value is null.

The following table shows the first data row that is inserted into **new_stock** from this INSERT statement.

Column	Value
col1	1
col2	baseball gloves
col3	SMT
col4	null
col5	case

Column	Value
col6	autographed

Character-position form of the FILE and INSERT statements

The FILE and INSERT statements that define information for the **dbload** utility can appear in a character-position form.

The examples in this topic are based on an input data file, **cust_loc_data**, which contains the last four columns (**city**, **state**, **zipcode**, and **phone**) of the **customer** table. Fields in the input file are padded with blanks to create data rows in which the location of data fields and the number of characters are the same across all rows. The definitions for these fields are CHAR(15), CHAR(2), CHAR(5), and CHAR(12), respectively. [Figure 2: A Sample Data File on page 91](#) displays the character positions and five example data rows from the **cust_loc_data** file.

Figure 2. A Sample Data File

	12	3
	1234567890123456789012345678901234	
Sunnyvale	CA94086408-789-8075	
Denver	CO80219303-936-7731	
Blue Island	NY60406312-944-5691	
Brighton	MA02135617-232-4159	
Tempe	AZ85253xxx-xxx-xxxx	

The following example of a **dbload** command file illustrates the character-position form of the FILE and INSERT statements. The example includes two new tables, **cust_address** and **cust_sort**, to receive the data. For the purpose of this example, **cust_address** contains four columns, the second of which is omitted from the column list. The **cust_sort** table contains two columns.

```
FILE cust_loc_data
  (city 1-15,
   state 16-17,
   area_cd 23-25 NULL = 'xxx',
   phone 23-34 NULL = 'xxx-xxx-xxxx',
   zip 18-22,
   state_area 16-17 : 23-25);
INSERT INTO cust_address (col1, col3, col4)
  VALUES (city, state, zip);
INSERT INTO cust_sort
  VALUES (area_cd, zip);
```

Syntax for the character-position form

The syntax for the character-position form specifies information that includes the character position within a data row that starts a range of character positions and the character position that ends a range of character positions.

The following diagram shows the syntax of the character-position FILE statement.

```

FILEfilename(
,fieldn
:start-end
NULL='null string'
)
    
```

Element	Purpose	Key Considerations
<i>-end</i>	Indicates the character position within a data row that ends a range of character positions	A hyphen must precede the <i>end</i> value.
<i>fieldn</i>	Assigns a name to the data field that you are defining with the range of character positions	None.
<i>filename</i>	Specifies the name of the input file	None.
<i>null string</i>	Specifies the data value for which dbload must substitute a null value	Must be a quoted string.
<i>start</i>	Indicates the character position within a data row that starts a range of character positions. If you specify <i>start</i> without <i>end</i> , it represents a single character.	None.

You can repeat the same character position in a data-field definition or in different fields.

The *null string* scope of reference is the data field for which you define it. You can define an explicit null string for each field that allows null entries.

Inserted data types correspond to the explicit or default column list. If the data-field width is different from its corresponding character column, inserted values are padded with blanks if the column is wider, or inserted values are truncated if the field is wider.

If the number of columns named is fewer than the number of columns in the table, **dbload** inserts the default value that is specified for the unnamed columns. If no default value is specified, **dbload** attempts to insert a null value. If the attempt violates a not-null restriction or a unique constraint, the insert fails, and an error message is returned.

If the INSERT statement omits the column names, the default INSERT specification is every column in the named table. If the INSERT statement omits the VALUES clause, the default INSERT specification is every field of the previous FILE statement.

An error results if the number of column names listed (or implied by default) does not match the number of values listed (or implied by default).

The syntax of **dbload** INSERT statements resembles INSERT statements in SQL, except that in **dbload**, INSERT statements cannot incorporate SELECT statements. The following diagram shows the syntax of the **dbload** INSERT statement for character-position form.

INSERT INTO

```
owner.
table
(
,column
)
VALUES clause 5
;
```

Element	Purpose	Key Considerations
<i>column</i>	Specifies the column that receives the new data	None.
<i>owner.</i>	Specifies the user name of the table owner	None.
<i>table</i>	Specifies the table that receives the new data	None.

The syntax for character-position form is identical to the syntax for delimiter form.

The user who runs **dbload** with this command file must have the Insert privilege on the named table.

How to write a dbload command file in character-position form

Command files must define data fields and use character positions to define the length of each field.

The FILE statement in the following example defines six data fields from the **cust_loc_data** table data rows.

```
FILE cust_loc_data
(city 1-15,
state 16-17,
area_cd 23-25 NULL = 'xxx',
phone 23-34 NULL = 'xxx-xxx-xxxx',
zip 18-22,
state_area 16-17 : 23-25);
INSERT INTO cust_address (col1, col3, col4)
VALUES (city, state, zip);
```

The statement names the fields and uses character positions to define the length of each field. Compare the FILE statement in the preceding example with the data rows in the following figure.

Figure 3. A Sample Data File

123 1234567890123456789012345678901234	
Sunnyvale++++++CA94086408-789-8075	— Data row 1
Tempe++++++AZ85253xxx-xxx-xxxx	— Data row 2

The FILE statement defines the following data fields, which are derived from the data rows in the sample data file.

5. See [VALUES Clause on page](#) .

Column	Values from Data Row 1	Values from Data Row 2
city	Sunnyvale++++++	Tempe+++++++
state	CA	AZ
area_cd	408	null
phone	408-789-8075	null
zip	94086	85253
state_area	CA408	AZxxx

The null strings that are defined for the **phone** and **area_cd** fields generate the null values in those columns, but they do not affect the values that are stored in the **state_area** column.

The INSERT statement uses the field names and values that are derived from the FILE statement as the value-list input. Consider the following INSERT statement:

```
INSERT INTO cust_address (col1, col3, col4)
VALUES (city, state, zip);
```

The INSERT statement uses the data in the sample data file and the FILE statement to put the following information into the **cust_address** table.

Column	Values from Data Row 1	Values from Data Row 2
col1	Sunnyvale++++++	Tempe+++++++
col2	null	null
col3	CA	AZ
col4	94086	85253

Because the second column (**col2**) in **cust_address** is not named, the new data row contains a null (assuming that the column permits nulls).

Consider the following INSERT statement:

```
INSERT INTO cust_sort
VALUES (area_cd, zip);
```

This INSERT statement inserts the following data rows into the **cust_sort** table.

Column	Values from Data Row 1	Values from Data Row 2
col1	408	null
col2	94086	85253

Because no column list is provided, **dbload** reads the names of all the columns in **cust_sort** from the system catalog. (You cannot insert data into a temporary table because temporary tables are not entered into the system catalog.) Field names

from the previous FILE statement specify the values to load into each column. You do not need one FILE statement for each INSERT statement.

Command file to load complex data types

You can create **dbload** command files that load columns containing complex data types into tables.

You can use **dbload** with the following data types:

- A BLOB or CLOB
- A SET inside a ROW type

The **dbload** utility does not work with the following data types:

- A CLOB or BLOB inside a ROW type
- A ROW type inside a SET



Important: All the load utilities (**dbexport**, **dbimport**, **dbload**, **onload**, **onunload**, and **onxfer**) rely on an export and import function. If you do not define this function when you write a user-defined data type, you cannot use these utilities.

Loading a new data type inside another data type can cause problems if the representation of the data contains handles. If a string represents the data, you must be able to load it.

You can use **dbload** with named row types, unnamed row types, sets, and lists.

Using the dbload utility with named row types

The procedure for using the **dbload** utility with named row types is somewhat different than the procedure for using **dbload** with other complex data types, because named row types are actually user-defined data types.

Suppose you have a table named **person** that contains one column with a named row type. Also suppose that the **person_t** named row type contains six fields: **name**, **address**, **city**, **state**, **zip**, and **bdate**.

The following syntax shows how to create the named row type and the table used in this example:

```
CREATE ROW TYPE person_t
(
  name VARCHAR(30) NOT NULL,
  address VARCHAR(20),
  city VARCHAR(20),
  state CHAR(2),
  zip VARCHAR(9),
  bdate DATE
);
CREATE TABLE person OF TYPE person_t;
```

To load data for a named row type (or for any user-defined data type)

1. Use the UNLOAD statement to unload the table to an input file. In this example, the input file sees the named row type as six separate fields:

```
Brown, James|13 First St.|San Francisco|CA|94070|01/04/1940|
Karen Smith|1820 Elm Ave #100|Fremont|CA|94502|01/13/1983|
```

2. Use the **dbschema** utility to capture the schema of the table and the row type. You must use the **dbschema -u** option to pick up the named row type.

```
dbschema -d stores_demo -u person_t > schema.sql
dbschema -d stores_demo -t person > schema.sql
```

3. Use DB-Access to re-create the **person** table in the new database.

For detailed steps, see [Use dbschema output as DB-Access input on page 114](#).

4. Create the **dbload** command file. This **dbload** command file inserts two rows into the **person** table in the new database.

```
FILE person.unl DELIMITER '|' 6;
INSERT INTO person;
```

This **dbload** example shows how to insert new data rows into the **person** table. The number of rows in the INSERT statement and the **dbload** command file must match:

```
FILE person.unl DELIMITER '|' 6;
INSERT INTO person
VALUES ('Jones, Richard', '95 East Ave.',
       'Philadelphia', 'PA',
       '19115',
       '03/15/97');
```

5. Run the **dbload** command:

```
dbload -d newdb -c uds_command -l errlog
```



Tip: To find the number of fields in an unloaded table that contains a named row type, count the number of fields between each vertical bar (|) delimiter.

Using the dbload utility with unnamed row types

You can use the **dbload** utility with unnamed row types, which are created with the ROW constructor and define the type of a column or field.

In the following example, the **devtest** table contains two columns with unnamed row types, **s_name** and **s_address**. The **s_name** column contains three fields: **f_name**, **m_init**, and **l_name**. The **s_address** column contains four fields: **street**, **city**, **state**, and **zip**.

```
CREATE TABLE devtest
(
s_name ROW(f_name varchar(20), m_init char(1), l_name varchar(20)
not null),
s_address ROW(street varchar(20), city varchar(20), state char(20),
zip varchar(9)
);
```

The data from the **devtest** table is unloaded into the **devtest.unl** file. Each data row contains two delimited fields, one for each unnamed row type. The ROW constructor precedes each unnamed row type, as follows:

```
ROW('Jim','K','Johnson')|ROW('10 Grove St.','Eldorado','CA','94108')|
ROW('Maria','E','Martinez')|ROW('2387 West Wilton
Ave.','Hershey','PA','17033')|
```

This **dbload** example shows how to insert data that contains unnamed row types into the **devtest** table. Put double quotes around each unnamed row type or the insert will not work.

```
FILE devtest.unl DELIMITER '|' 2;
INSERT INTO devtest (s_name, s_address)
VALUES ("row('Stephen', 'M', 'Wu')",
"row('1200 Grand Ave.', 'Richmond', 'OR', '97200')");
```

Using the dbload utility with collection data types

You can use the **dbload** utility with collection data types such as SET, LIST, and MULTISET.

SET data type example

The SET data type is an unordered collection type that stores unique elements. The number of elements in a SET data type can vary, but no nulls are allowed.

The following statement creates a table in which the **children** column is defined as a SET:

```
CREATE TABLE employee
(
    name char(30),
    address char(40),
    children SET (varchar(30) NOT NULL)
);
```

The data from the **employee** table is unloaded into the **employee.unl** file. Each data row contains four delimited fields. The first set contains three elements (**Karen**, **Lauren**, and **Andrea**), whereas the second set contains four elements. The SET constructor precedes each SET data row.

```
Muriel|5555 SW Merry
Sailing Dr.|02/06/1926|SET{'Karen','Lauren','Andrea'}|
Larry|1234 Indian Lane|07/31/1927|SET{'Martha',
'Melissa','Craig','Larry'}|
```

This **dbload** example shows how to insert data that contains SET data types into the **employee** table in the new database. Put double quotes around each SET data type or the insert does not work.

```
FILE employee.unl DELIMITER '|' 4;
INSERT INTO employee
VALUES ('Marvin', '10734 Pardee', '06/17/27',
"SET{'Joe', 'Ann'}");
```

LIST data type example

The LIST data type is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values.

The following statement creates a table in which the **month_sales** column is defined as a LIST:

```
CREATE TABLE sales_person
(
  name CHAR(30),
  month_sales LIST(MONEY NOT NULL)
);
```

The data from the **sales_person** table is unloaded into the **sales.unl** file. Each data row contains two delimited fields, as follows:

```
Jane Doe|LIST{'4.00', '20.45', '000.99'}|
Big Earner|LIST{'0000.00', '00000.00', '999.99'}|
```

This **dbload** example shows how to insert data that contains LIST data types into the **sales_person** table in the new database. Put double quotes around each LIST data type or the insert does not work.

```
FILE sales_person.unl DELIMITER '|' 2;
INSERT INTO sales_person
VALUES ('Jenny Chow', "{587900, 600000}");
```

You can load multisets in a similar manner.

The dbschema utility

The **dbschema** utility displays the SQL statements (the *schema*) that are necessary to replicate database objects.

You can also use the **dbschema** utility for the following purposes:

- To display the distributions that the UPDATE STATISTICS statement creates.
- To display the schema for the Information Schema views
- To display the schema for creating objects such as databases, tables, sequences, synonyms, storage spaces, chunks, logs, roles, and privileges
- To display the distribution information that is stored for one or more tables in the database
- To display information about user-defined data types and row types

After you obtain the schema of a database, you can redirect the **dbschema** output to a file that you can use with DB-Access.

The **dbschema** utility is supported on all updatable secondary servers.

The **dbschema** utility is also supported on read-only secondary servers. However, the **dbschema** utility displays a warning message when running on these servers.



Attention: Use of the **dbschema** utility can increment sequence objects in the database, creating gaps in the generated numbers that might not be expected in applications that require serialized integers.

Related information

[Data-migration tools on page 2](#)

[Choosing a tool for moving data before migrating between operating systems on page 62](#)

Object modes and violation detection in dbschema output

The output from the **dbschema** utility shows object modes and supports violation detection.

The **dbschema** output shows:

- The names of not-null constraints after the not-null specifications.

You can use the output of the utility as input to create another database. If the same names were not used for not-null constraints in both databases, problems could result.

- The object mode of objects that are in the disabled state. These objects can be constraints, triggers, or indexes.
- The object mode of objects that are in the filtering state. These objects can be constraints or unique indexes.
- The violations and diagnostics tables that are associated with a base table (if violations and diagnostics tables were started for the base table).

For more information about object modes and violation detection, see the SET, START VIOLATIONS TABLE, and STOP VIOLATIONS TABLE statements in the *Informix® Guide to SQL: Syntax*.

Guidelines for using the dbschema utility

You can use delimited identifiers with the **dbschema** utility. The **dbschema** utility detects database objects that are keywords, mixed case, or that have special characters, and the utility places double quotation marks around those keywords.



Global Language Support: You must disable SELECT triggers and correctly set GLS environment variables before using the **dbschema** utility.

When the GLS environment variables are set correctly, as the *Informix® GLS User's Guide* describes, the **dbschema** utility can handle foreign characters.

Syntax of the dbschema command

The **dbschema** command displays the SQL statements (the *schema*) that are necessary to replicate a specified database object. The command also shows the distributions that the UPDATE STATISTICS statement creates.

dbschema

Table options Database options

UDT options

-V

-version

Storage, space, and log options

No owner option

UDT options

6

-u all

-ua -ui

udt_name

Table options

Tables, Views, or Procedures⁷

Synonyms⁸

Privileges⁹

-hd

all

Table Name¹⁰

-r

roleall

11

Database options

-ss

-seq

sequenceall

-ddatabase

-wpassword

Storage space and log options

-c

-ns

*file_name*¹²

No owner option

-nw

6. See [User-defined and complex data types on page 104](#)

7. See [Table, view, or procedure creation on page 106](#)

8. See [Synonym creation on page 106](#)

9. See [Privileges on page 110](#)

10. See [Identifier on page](#) .

11. See [Role creation on page 109](#)

12. See [Storage space, chunk, and log creation on page 107](#)

Element	Purpose	Additional Information
all	Directs dbschema to include all the tables or sequence objects in the database, or all the user-defined data types in the display of distributions.	None.
-c <i>file_name</i>	Generates commands to reproduce storage spaces, chunks, physical logs, and logical logs.	If you use the -c element without the -ns element, the database server generates SQL administration API commands. If you use the -c element and also use the -ns element, the database server generates onspaces or onparams commands.
-d <i>database</i>	Specifies the database to which the schema applies. The <i>database</i> can be on a remote database server.	To use more than the simple name of the database, see Database Name on page .
<i>filename</i>	Specifies the name of the file that contains the dbschema output.	If you omit a file name, dbschema sends the output to the screen. If you specify a file name, dbschema creates a file named <i>filename</i> to contain the dbschema output.
-hd	Displays the distribution as data values.	If you specify the ALL keyword for the table name, the distributions for all the tables in the database are displayed.
-it	Sets the isolation type for dbschema while dbschema queries catalog tables. Isolation types are: DR = Dirty Read CR = Committed Read CS = Cursor Stability CRU = Committed Read with RETAIN UPDATE LOCKS CSU = Cursor Stability with RETAIN UPDATE LOCKS DRU = Dirty Read with RETAIN UPDATE LOCKS LC = Committed Read, Last Committed RR = Repeatable Read	
-l	Sets the lock mode to wait <i>number of</i> seconds for dbschema while dbschema queries catalog tables.	None.

Element	Purpose	Additional Information
-ns	Generates onspaces or onparams utility commands to reproduce storage spaces, chunks, physical logs, and logical logs.	The -c element must precede the -ns element in your command.
-nw	Generates the SQL for creating an object without the specification of an owner.	The -nw element is also a dbexport command option.
-q	Suppresses the database version from the header.	This optional element precedes other elements.
-r	Generates information about the creation of roles.	For details, see Role creation on page 109 .
-seq <i>sequence</i>	Generates the DDL statement to define the specified <i>sequence</i> object	None.
-ss	Generates server-specific information	This option is ignored if no table schema is generated.
-si	Excludes the generation of index storage clauses for non-fragmented tables	This option is available only with the -ss option.
-sl <i>length</i>	Specifies the maximum length, in bytes, of unformatted CREATE TABLE and ALTER TABLE statements.	None.
-u all	Prints the definitions of all user-defined data types, including all functions and casts defined over the types.	Specify -u all to include all the user-defined data types in the list of distributions.
-ua <i>udt_name</i>	Prints the definition of a user-defined data type, including functions and casts defined over an opaque or constructor type.	None.
-ui <i>udt_name</i>	Prints the definition of a user-defined data type, including type inheritance.	None.
-V	Displays the software version number and the serial number	None.
-version	Extends the -V option to display additional information about the build version, host, operating system, build number and date, and the GLS version.	None.
-w <i>password</i>	Specifies the database password, if you have one.	None.

You must be the DBA or have the Connect or Resource privilege for the database before you can run **dbschema** on it.

Example**Example**

The following command generates the schema with all the tables or sequence objects in the `customer` database, but without the specification of an owner:

```
dbschema -d customer all -nw
```

Database schema creation

You can create the schema for an entire database or for a portion of the database.

Use the **dbschema** utility options to perform the following actions:

- Display CREATE SYNONYM statements by owner, for a specific table or for the entire database.
- Display the CREATE TABLE, CREATE VIEW, CREATE FUNCTION, or CREATE PROCEDURE statement for a specific table or for the entire database.
- Display all GRANT privilege statements that affect a specified user or that affect all users for a database or a specific table. The user can be either a user name or role name.
- Display user-defined and row data types with or without type inheritance.
- Display the CREATE SEQUENCE statement defining the specified *sequence* object, or defining all sequence objects in the database.

When you use **dbschema** and specify only the database name, it is equivalent to using **dbschema** with all its options (except for the **-hd** and **-ss** options). In addition, if Information Schema views were created for the database, this schema is shown.

For example, the following two commands are equivalent:

```
dbschema -d stores_demo
dbschema -s all -p all -t all -f all -d stores_demo
```

SERIAL fields included in CREATE TABLE statements that **dbschema** displays do not specify a starting value. New SERIAL fields created with the schema file have a starting value of `1`, regardless of their starting value in the original database. If this value is not acceptable, you must modify the schema file.

Creating schemas for databases across a UNIX™ or Linux™ network

The **dbschema -d** option creates and displays the schema for databases on a UNIX™ or Linux™ network.

You can specify a database on any accessible database server.

The following command displays the schema for the **stores_demo** database on the **finland** database server on the UNIX™ or Linux™ system console:

```
dbschema -d //finland/stores_demo
```

Changing the owner of an object

You can edit **dbschema** output to change the owner of a new object.

The **dbschema** utility uses the *owner.object* convention when it generates any CREATE TABLE, CREATE INDEX, CREATE SYNONYM, CREATE VIEW, CREATE SEQUENCE, CREATE PROCEDURE, CREATE FUNCTION, or GRANT statement, and when it reproduces any unique, referential, or check constraint. As a result, if you use the **dbschema** output to create a new object (table, index, view, procedure, constraint, sequence, or synonym), the owner of the original object owns the new object. If you want to change the owner of the new object, you must edit the **dbschema** output before you run it as an SQL script.

You can use the output of **dbschema** to create a new function if you also specify the path name to a file in which compile-time warnings are stored. This path name is displayed in the **dbschema** output.

For more information about the CREATE TABLE, CREATE INDEX, CREATE SYNONYM, CREATE VIEW, CREATE SEQUENCE, CREATE PROCEDURE, CREATE FUNCTION, and GRANT statements, see the *Informix® Guide to SQL: Syntax*.

dbschema server-specific information

The **dbschema -ss** option generates server-specific information. The **-ss** option always generates the lock mode, extent sizes, and the dbspace name if the dbspace name is different from the database dbspace. In addition, if tables are fragmented, the **-ss** option displays information about the fragmentation strategy.

When you specify the **dbschema -ss** option, the output also displays any GRANT FRAGMENT statements that are issued for a particular user or in the entire schema.

The **-si option**, which is available only with the **-ss** option, excludes the generation of index storage clauses for non-fragmented tables.

If the dbspace contains multiple partitions, dbspace partition names appear in the output.

For information about fragment-level authority, see the GRANT FRAGMENT and REVOKE FRAGMENT statements in the *Informix® Guide to SQL: Syntax*.

User-defined and complex data types

The **dbschema -u** option displays the definitions of any user-defined and complex data types that the database contains. The suboption **i** adds the type inheritance to the information that the **dbschema -u** option displays.

The following command displays all the user-defined and complex data types for the **stork** database:

```
dbschema -d stork -u all
```

Output from **dbschema** that ran with the specified option `-u all` might appear as the following example shows:

```
create row type 'informix'.person_t
(
  name varchar(30, 10) not null,
  address varchar(20, 10),
  city varchar(20, 10),
  state char(2),
  zip integer,
```

```

bdate date
);
create row type 'informix'.employee_t
(
  salary integer,
  manager varchar(30, 10)
) under person_t;

```

The following command displays the user-defined and complex data types, as well as their type inheritance for the **person_t** table in the **stork** database:

```
dbschema -d stork -ui person_t
```

Output from **dbschema** that ran with the option `-ui person_t` might appear as the following example shows:

```

create row type 'informix'.person_t
(
  name varchar(30, 10) not null,
  address varchar(20, 10),
  city varchar(20, 10),
  state char(2),
  zip integer,
  bdate date
);
create row type 'informix'.employee_t
(
  salary integer,
  manager varchar(30, 10)
) under person_t;
create row type 'informix'.sales_rep_t
(
  rep_num integer,
  region_num integer,
  commission decimal(16),
  home_office boolean
) under employee_t;

```

Sequence creation

The `dbschema -seq sequence` command generates information about sequence creation.

The following syntax diagram fragment shows sequence creation.

-seq

sequence **all**

Element	Purpose	Key Considerations
-seq <i>sequence</i>	Displays the CREATE SEQUENCE statement defining <i>sequence</i>	None.
-seq all	Displays all CREATE SEQUENCE statements for the database	None.

Running **dbschema** with option **-seq** sequitur might produce this output:

```
CREATE SEQUENCE sequitur INCREMENT 10 START 100 NOCACHE CYCLE
```

For more information about the CREATE SEQUENCE statement, see the *Informix® Guide to SQL: Syntax*.

Synonym creation

The `dbschema -s` command generates information about synonym creation.

The following syntax diagram fragment shows the creation of synonyms.

Synonyms

-s

ownername **all**

Element	Purpose	Key Considerations
-s <i>ownername</i>	Displays the CREATE SYNONYM statements owned by <i>ownername</i>	None.
-s all	Displays all CREATE SYNONYM statements for the database, table, or view specified	None.

Output from **dbschema** that ran with the specified option `-s alice` might appear as the following example shows:

```
CREATE SYNONYM 'alice'.cust FOR 'alice'.customer
```

For more information about the CREATE SYNONYM statement, see the *Informix® Guide to SQL: Syntax*.

Table, view, or procedure creation

Several **dbschema** options generate information that shows the creation of tables, views, and procedures.

The following syntax diagram shows the creation of tables, views, and procedures.

Tables, Views, or Procedures:

-t

table view **all**

-t

table view

p f

all

all

Element	Purpose	Key Considerations
-f all	Limits the SQL statement output to those statements that replicate all functions and procedures	None.

Element	Purpose	Key Considerations
-f function	Limits the SQL statement output to only those statements that replicate the specified function	None.
-f procedure	Limits the SQL statement output to only those statements that replicate the specified procedure	None.
-ff all	Limits the SQL statement output to those statements that replicate all functions	None.
-fp all	Limits the SQL statement output to those statements that replicate all procedures	None.
-t table	Limits the SQL statement output to only those statements that replicate the specified table	None.
-t view	Limits the SQL statement output to only those statements that replicate the specified view	None.
-t all	Includes in the SQL statement output all statements that replicate all tables and views	None.

For more information about the CREATE PROCEDURE and CREATE FUNCTION statements, see the *Informix® Guide to SQL: Syntax*.

Storage space, chunk, and log creation

The `dbschema -c` command generates SQL administration API commands for reproducing storage spaces, chunks, logical logs, and physical logs. If you use the `dbschema -c -ns` command, the database server generates **onspaces** or **onparams** utility commands for reproducing storage spaces, chunks, physical logs, and logical logs.

For example:

- Run the following command to generate a file named `dbschema1.out` that contains the commands for reproducing the storage spaces, chunks, physical logs, and logical logs in SQL Admin API format:

```
dbschema -c dbschema1.out
```

- Run the following command to generate a file named `dbschema2.out` that contains the commands for reproducing the storage spaces, chunks, physical logs, and logical logs in **onspaces** and **onparams** utility format:

```
dbschema -c -ns dbschema2.out
```

Optionally, specify **-q** before you specify `-c` or `-c -ns` to suppress the database version when you run the command. For example, specify:

```
dbschema -q -c -ns dbschema3.out
```

Sample output for the creation of storage spaces, chunks, and logs

The output of the **dbschema -c** or **dbschema -c -ns** commands contain all of the SQL administration API or **onspaces** and **onparams** utility commands that you can use to reproduce storage spaces, chunks, and logs.

Example of output in SQL administration API format

```
# Dbspace 1 -- Chunk 1
EXECUTE FUNCTION TASK ('create dbspace', 'rootdbs',
  '/export/home/informix/data/rootdbs1150fc4', '200000',
  '0', '2', '500', '100')

# Dbspace 2 -- Chunk 2
EXECUTE FUNCTION TASK ('create dbspace', 'datadbs1',
  '/export/home/informix/data/datadbs1150fc4', '5000000',
  '0', '2', '100', '100')

# Dbspace 3 -- Chunk 3
EXECUTE FUNCTION TASK ('create dbspace', 'datadbs2',
  '/export/home/informix/data/datadbs2150fc4', '5000000',
  '0', '2', '100', '100')

# Dbspace 4 -- Chunk 4
EXECUTE FUNCTION TASK ('create dbspace', 'datadbs3',
  '/export/home/informix/data/datadbs3_1150fc4', '80000',
  '16', '8', '400', '400')
EXECUTE FUNCTION TASK ('start mirror', 'datadbs3',
  '/export/home/informix/data/datadbs3_1150fc4', '80000',
  '16', '/export/home/informix/data/mdatadbs3_1150fc4', '16')

# Dbspace 5 -- Chunk 5
EXECUTE FUNCTION TASK ('create tempdbspace', 'tempdbs',
  '/export/home/informix/data/tempdbs_1150fc4', '1000',
  '0', '2', '100', '100')

# Dbspace 6 -- Chunk 6
EXECUTE FUNCTION TASK ('create sbspace', 'sbspace',
  '/export/home/informix/data/sbspace_1150fc4',
  '1000', '0')

# Dbspace 6 -- Chunk 7
EXECUTE FUNCTION TASK ('add chunk', 'sbspace',
  '/export/home/informix/data/sbspace_1_1150fc4',
  '1000', '0')

# Dbspace 7 -- Chunk 8
EXECUTE FUNCTION TASK ('create blobdbspace', 'blobdbs',
  '/export/home/informix/data/blobdbs_1150fc4',
  '1000', '0', '4')

# External Space 1
EXECUTE FUNCTION TASK ('create extspace', 'extspace',
  '/export/home/informix/data/extspac_1150fc4')

# Physical Log
EXECUTE FUNCTION TASK ('alter plog', 'rootdbs', '60000')
```

```
# Logical Log 1
EXECUTE FUNCTION TASK ('add log', 'rootdbs', '10000')
```

Example of output in onspaces and onparams utility format

```
# Dbspace 1 -- Chunk 1
onspaces -c -d rootdbs -k 2 -p
/export/home/informix/data/rootdbs1150fc4
-o 0 -s 200000 -en 500 -ef 100

# Dbspace 2 -- Chunk 2
onspaces -c -d datadbs1 -k 2 -p
/export/home/informix/data/datadbs1150fc4
-o 0 -s 5000000 -en 100 -ef 100

# Dbspace 3 -- Chunk 3
onspaces -c -d datadbs2 -k 2 -p
/export/home/informix/data/datadbs2150fc4
-o 0 -s 5000000 -en 100 -ef 100

Dbspace 4 -- Chunk 4
onspaces -c -d datadbs3 -k 8
-p /export/home/informix/data/datadbs3_1150fc4
-o 16 -s 80000 -en 400 -ef 400
-m /export/home/informix/data/mdatadbs3_1150fc4 16

# Dbspace 5 -- Chunk 5
onspaces -c -d tempdbs -k 2 -t -p
/export/home/informix/data/tempdbs_1150fc4 -o 0 -s 1000

# Dbspace 6 -- Chunk 6
onspaces -c -S sbspace -p
/export/home/informix/data/sbspace_1150fc4
-o 0 -s 1000 -Ms 500

# Dbspace 7 -- Chunk 7
onspaces -c -b blobdbs -g 4 -p
/export/home/informix/data/blobdbs_1150fc4 -o 0 -s 1000

# External Space 1
onspaces -c -x extspace -l
/export/home/informix/data/extspac_1150fc4

# Logical Log 1
onparams -a -d rootdbs -s 10000
```

Role creation

The `dbschema -r` command generates information on the creation of roles.

The following syntax diagram shows the creation of roles.

Roles

-r

role all

Element	Purpose	Key Considerations
-r role	Displays the CREATE ROLE and GRANT statements that are needed to replicate and grant the specified role.	You cannot specify a list of users or roles with the -r option. You can specify either one role or all roles.
-r all	Displays all CREATE ROLE and GRANT statements that are needed to replicate and grant all roles.	None

The following **dbschema** command and output show that the role **calen** was created and was granted to **cathl**, **judith**, and **sallyc**:

```
sharky% dbschema -r calen -d stores_demo

DBSCHEMA Schema Utility
Software Serial Number RDS#N0000000
create role calen;

grant calen to cathl with grant option;
grant calen to judith ;
grant calen to sallyc ;
```

Privileges

The **dbschema -p** command generates information on privileges.

The following syntax diagram fragment shows privileges information.

Privileges

-p

user all

Element	Purpose	Key Considerations
-p user	Displays the GRANT statements that grant privileges to <i>user</i> , where <i>user</i> is a user name or role name. Specify only one user or role	You cannot specify a specific list of users with the -p option. You can specify either one user or role, or all users and roles.
-p all	Displays the GRANT statements for all users for the database, table, or view specified, or to all roles for the table specified	None.

The output also displays any GRANT FRAGMENT statements that are issued for a specified user or role or (with the **all** option) for the entire schema.

Granting privileges

You can generate **dbschema** information about the grantor of a GRANT statement.

In the **dbschema** output, the AS keyword indicates the grantor of a GRANT statement. The following example output indicates that **norma** issued the GRANT statement:

```
GRANT ALL ON 'tom'.customer TO 'claire' AS 'norma'
```

When the GRANT and AS keywords appear in the **dbschema** output, you might need to grant privileges before you run the **dbschema** output as an SQL script. Referring to the previous example output line, the following conditions must be true before you can run the statement as part of a script:

- User **norma** must have the Connect privilege to the database.
- User **norma** must have all privileges WITH GRANT OPTION for the table **tom.customer**.

For more information about the GRANT, GRANT FRAGMENT, and REVOKE FRAGMENT statements, see the *Informix® Guide to SQL: Syntax*.

Displaying privilege information for a role

You can generate **dbschema** information about the privileges that were granted for a particular role.

A *role* is a classification with privileges on database objects granted to the role. The DBA can assign the privileges of a related work task, such as an engineer, to a role and then grant that role to users, instead of granting the same set of privileges to every user. After a role is created, the DBA can use the GRANT statement to grant the role to users or to other roles.

For example, issue the following **dbschema** command and to display privileges that were granted for the **calen** role.

```
sharky% dbschema -p calen -d stores_demo
```

An example of information the **dbschema** utility displays is:

```
grant alter on table1 to 'calen'
```

Distribution information for tables in dbschema output

The **dbschema -hd** command with the name of the table retrieves the distribution information that is stored for a table in a database. If you specify the ALL keyword for the table name, the distributions for all the tables in the database are displayed.

During the **dbimport** operation, distribution information is created automatically for leading indexes on non-opaque columns. Run the UPDATE STATISTICS statement in MEDIUM or HIGH mode to create distribution information about tables that have the following types of indexes:

- Virtual Index Interface (VII) or function indexes
- Indexes on columns of user-defined data types
- Indexes on columns of built-in opaque data types (such as BOOLEAN or LVARCHAR)

Output from the **dbschema** utility shows distribution information if you used the SAMPLING SIZE keywords when UPDATE STATISTICS in MEDIUM or HIGH mode ran on the table.

For information about using the UPDATE STATISTICS statement, see the *Informix® Guide to SQL: Syntax*.

The output of **dbschema** for distributions is provided in the following parts:

- Distribution description
- Distribution information
- Overflow information

Each section of **dbschema** output is explained in the following sections. As an example, the discussion uses the following distribution for the fictional table called **invoices**. This table contains 165 rows, including duplicates.

You can generate the output for this discussion with a call to **dbschema** that is similar to the following example:

```
dbschema -hd invoices -d pubs_stores_demo
```

Example of dbschema output showing distribution information

The **dbschema** output can show the data distributions that have been created for the specified table and the date when the UPDATE STATISTICS statement that generated the distributions ran.

The follow example of **dbschema** output shows distribution information.

```
Distribution for cathl.invoices.invoice_num

High Mode, 10.000000 Resolution

--- DISTRIBUTION ---

      (              5)
1: ( 16,      7,      11)
2: ( 16,      6,      17)
3: ( 16,      8,      25)
4: ( 16,      8,      38)
5: ( 16,      7,      52)
6: ( 16,      8,      73)
7: ( 16,     12,      95)
8: ( 16,     12,     139)
9: ( 16,     11,     182)
10: ( 10,      5,     200)

--- OVERFLOW ---

1: (  5,              56)
2: (  6,              63)
}
```

Description of the distribution information in the example

The first part of the sample **dbschema** output describes which data distributions have been created for the specified table.

The name of the table is stated in the following example:

```
Distribution for cathl.invoices.invoice_num
```

The output is for the **invoices** table, which is owned by user `cathl`. This data distribution describes the column **invoice_num**. If a table has distributions that are built on more than one column, **dbschema** lists the distributions for each column separately.

The **dbschema** output also shows the date when the UPDATE STATISTICS statement that generated the distributions ran. You can use this date to tell how outdated your distributions are.

The last line of the description portion of the output describes the mode (MEDIUM or HIGH) in which the distributions were created, and the resolution. If you create the distributions with medium mode, the confidence of the sample is also listed. For example, if the UPDATE STATISTICS statement runs in HIGH mode with a resolution of `10`, the last line appears as the following example shows:

```
High Mode, 10.000000 Resolution
```

Distribution information in dbschema output

The distribution information in **dbschema** output describes the bins that are created for the distribution, the range of values in the table and in each bin, and the number of distinct values in each bin.

Consider the following example:

```
(
 5)
1: ( 16, 7, 11)
2: ( 16, 6, 17)
3: ( 16, 8, 25)
4: ( 16, 8, 38)
5: ( 16, 7, 52)
6: ( 16, 8, 73)
7: ( 16, 12, 95)
8: ( 16, 12, 139)
9: ( 16, 11, 182)
10: ( 10, 5, 200)
```

The first value in the rightmost column is the smallest value in this column. In this example, it is `5`.

The column on the left shows the bin number, in this case `1` through `10`. The first number in parentheses shows how many values are in the bin. For this table, 10 percent of the total number of rows (`165`) is rounded down to `16`. The first number is the same for all the bins except for the last. The last row might have a smaller value, indicating that it does not have as many row values. In this example, all the bins contain 16 rows except the last one, which contains 10.

The middle column within the parentheses indicates how many distinct values are contained in this bin. Thus, if there are 11 distinct values for a 16-value bin, it implies that one or more of those values are duplicated at least once.

The right column within the parentheses is the highest value in the bin. The highest value in the last bin is also the highest value in the table. For this example, the highest value in the last bin is `200`.

Overflow information in dbschema output

The last portion of the **dbschema** output shows values that have many duplicates.

The number of duplicates of indicated values must be greater than a critical amount that is determined as approximately 25 percent of the resolution times the number of rows. If left in the general distribution data, the duplicates would skew the distribution, so they are moved from the distribution to a separate list, as the following example shows:

```
--- OVERFLOW ---
1: ( 5,      56)
2: ( 6,      63)
```

For this example, the critical amount is $0.25 * 0.10 * 165$, or 4.125. Therefore, any value that is duplicated five or more times is listed in the overflow section. Two values in this distribution are duplicated five or more times in the table: the value 56 is duplicated five times, and the value 63 is duplicated six times.

Use dbschema output as DB-Access input

You can use the **dbschema** utility to get the schema of a database and redirect the **dbschema** output to a file. Later, you can import the file into DB-Access and use DB-Access to re-create the schema in a new database.

Inserting a table into a dbschema output file

You can insert CREATE TABLE statements into the **dbschema** output file and use this output as DB-Access input.

The following example copies the CREATE TABLE statements for the customer table into the **dbschema** output file, **tab.sql**:

```
dbschema -d db -t customer > tab.sql
```

Remove the header information about **dbschema** from the output file, **tab.sql**, and then use DB-Access to re-create the table in another database, as follows:

```
dbaccess db1 tab.sql
```

Re-creating the schema of a database

You can use **dbschema** and DB-Access to save the schema from a database and then re-create the schema in another database. A **dbschema** output file can contain the statements for creating an entire database.

About this task

To save a database schema and re-create the database:

1. Use **dbschema** to save the schema to an output file, such as **db.sql**:

```
dbschema -d db > db.sql
```

You can also use the **-ss** option to generate server-specific information:

```
dbschema -d db -ss > db.sql
```

2. Remove the header information about **dbschema**, if any, from the output file.
3. Add a CREATE DATABASE statement at the beginning of the output file or use DB-Access to create a new database.
4. Use DB-Access to re-create the schema in a new database:

```
dbaccess - db.sql
```

When you use **db.sql** to create a database on a different database server, confirm that dbspaces exist.

Results

The databases **db** and **testdb** differ in name but have the same schema.

The LOAD and UNLOAD statements

You can use the SQL LOAD and UNLOAD statements to move data. The LOAD statement is moderately fast and easy to use, but it only accepts specified data formats. You usually use the LOAD statement with data that is prepared with an UNLOAD statement.

You can use the UNLOAD statement in DB-Access to unload selected rows from a table into a text file.

The UNLOAD statement lets you manipulate the data as you unload it, but it requires that you unload to files on disk instead of to tape. If you unload to disk files, you might need to use UNIX™, Linux™, or Windows™ utilities to load those files onto tape.

To load tables, use LOAD or **dbload**. To manipulate a data file that you are loading or to access a database while it is loading, use the **dbload** utility. The cost of the flexibility is the time you spend creating the **dbload** command file and slower execution. When possible, use the LOAD statement, which is faster than **dbload**.

If the database contains label-based access control (LBAC) objects, you can load or unload only those rows in which your security label dominates the column-security label or the row-security label. If entire table is to be loaded or unloaded, you must have the necessary LBAC credentials for writing/reading all of the labeled rows and columns. For more information about LBAC objects, see the *Informix® Security Guide* and the *Informix® Guide to SQL: Syntax*.

Related information

[Data-migration tools on page 2](#)

[Choosing a tool for moving data before migrating between operating systems on page 62](#)

Syntax of the UNLOAD statement

The UNLOAD statement in DB-Access unloads selected rows from a table into a text file.

UNLOAD TO

' *filename* '

DELIMITER ' *delimiter* '

SELECT Statement ¹³

Element	Purpose	Key Considerations
<i>delimiter</i>	Character to use as delimiter	Requirements: See Syntax for the delimiter form on page 87
<i>filename</i>	Specifies the input file	None.

This syntax diagram is only for quick reference. For details about the syntax and use of the UNLOAD statement, see [UNLOAD statement on page](#)

Syntax of the LOAD statement

The LOAD statement in DB-Access appends rows to an existing table of a database.

LOAD FROM

' *filename* '

DELIMITER ' *delimiter* '

INSERT INTO

Table Name ¹⁴

Synonym Name ¹⁴

View Name ¹⁴

(
column
)

Element	Purpose	Key Considerations
<i>column</i>	The name of a column to receive data from <i>filename</i>	Must be a column in the specified table or view.
<i>delimiter</i>	Character to use as delimiter	See Syntax for the delimiter form on page 87
<i>filename</i>	Specifies the input file	None.

This syntax diagram is only for quick reference. For details about the syntax and use of the LOAD statement, see [LOAD statement on page](#)

13. See [SELECT statement on page](#)

14. See [Identifier on page](#)

Load and unload statements for locales that support multibyte code sets

For locales that support multibyte code sets, be sure that the declared size (in bytes) of any column that receives character data is large enough to store the entire data string.

For some locales, this can require up to 4 times the number of logical characters in the longest data string.

Load and unload operations for nondefault locales and GL_DATETIME or DBTIME environment variables

In nondefault locales, operations that load or unload DATETIME or INTERVAL values can be sensitive to the settings of the **GL_DATETIME**, **DBTIME**, and **USE_DTENV** environment variables.

If the database uses a nondefault locale and the **GL_DATETIME** or **DBTIME** environment variable has a nondefault setting, you must set the **USE_DTENV** environment variable to the value of `1` before you can process localized DATETIME or INTERVAL values correctly

- with the LOAD or UNLOAD statements of **DB-Access**,
- or with the **dbimport** or **dbexport** utilities,
- or in DML operations on objects that the CREATE EXTERNAL TABLE statement defined.

The onunload and onload utilities

The **onunload** and **onload** utilities provide the fastest way to move data between computers that use the same database server on the same platform.

For example, your site purchases a more powerful UNIX™ computer to allow faster access for users. You need to transfer existing databases to the new database server on the new computer. Use **onunload** to unload data from the first database server and then use **onload** to load the data into the second database server. Both database servers must have the same version number, or they must have compatible version numbers. You can move an entire database or selected tables only, but you cannot modify the database schema.

The **onunload** utility can unload data more quickly than either **dbexport** or the UNLOAD statement because **onunload** copies the data in binary format and in page-sized units. The **onload** utility takes a tape or a file that the **onunload** utility creates and re-creates the database or the table.

The **onunload** and **onload** utilities are faster than **dbimport**, **dbload**, or LOAD but are much less flexible and do not let you modify the database schema or move from one operating system or database server version to another.

Related information

[Data-migration tools on page 2](#)

[Choosing a tool for moving data before migrating between operating systems on page 62](#)

Guidelines for when to use the onunload and onload utilities

You can use **onunload** and **onload** only when certain conditions are met.

You can use only **onunload** and **onload** if your answer to each of the following questions is yes. If your answer is *no*, you cannot use **onunload** and **onload**.

Use onunload and onload only if your answer to each question is yes	My answer
Is the target database server on the same hardware platform?	
Do you want to move to another database server of the same version?	
Do you want to keep the existing database schema without modifying it?	
Do you want to move an entire database or an entire table?	
Are the page images compatible?	
Are the numeric representations the same?	

When you cannot use the onunload and onload utilities

Because the data is written in page-sized units, you cannot use **onunload** and **onload** to move data between UNIX™ or Linux™ and Windows™ because they use different page sizes. For example, the page size is 2 KB on some UNIX™ systems and 4 KB on Windows™.

Additionally, you cannot use **onunload** and **onload**:

- To move data between GLS and non-GLS databases.
- To move compressed data from one database to another.
You must uncompress data in compressed tables and fragments before you use the **onload** and **onunload** utilities.
- To move external tables or databases that contain external tables.
You must drop all the external tables before you use the **onunload** utility.
- To move tables and databases that contain extended or smart-large-object data types

Requirements for using the onload and onunload utilities

The **onload** and **onunload** utilities have limitations. You can use these utilities only to move data between database servers of the same version on the same operating system. You cannot modify the database schema, logging must be turned off, and the utilities can be difficult to use.

The **onload** and **onunload** utilities have the following requirements:

- The original database and the target database must be from the same version of the database server. You cannot use the **onload** and **onunload** utilities to move data from one version to another version.
- You cannot use **onload** and **onunload** to move data between different types of database servers.

- The **onload** command must have the same scope as the corresponding **onunload** command that unloaded the same table or tables that **onload** references. You cannot, for example, use **onunload** to unload an entire database, and then use **onload** to load only a subset of the tables from that database.
- Do not use **onload** and **onunload** to move data if the database contains extended or smart-large-object data types.
- Because the tape that **onload** reads contains binary data that is stored in disk-page-sized units, the computers where the original database resides (where you use **onunload**) and where the target database will reside (where you use **onload**) must have the same page size, the same representation of numeric data, the same byte alignment for structures and unions.
- You cannot use **onload** and **onunload** to move data between non-GLS and GLS locales.
- You cannot use **onload** and **onunload** on servers in high-availability clusters.
- You cannot use **onload** and **onunload** if you compressed tables or fragments.

You can use **onunload** and **onload** to move data between databases if the NLS and GLS locales are identical.

If the page sizes are different, **onload** fails. If the alignment or numeric data types on the two computers are different (for example, with the most significant byte as last instead of first, or different float-type representations), the contents of the data page could be misinterpreted.

How the onunload and onload utilities work

The **onunload** utility, which unloads data from a database, writes a database or table into a file on tape or disk. The **onload** utility loads data that was created with the **onunload** command into the database server.

The **onunload** utility unloads the data in binary form in disk-page units, making this utility more efficient than **dbexport**.

You can use the **onunload** utility to move data between computers that have the same version of the database server.



Important: You cannot use the **onload** and **onunload** utilities to move data from one version of a database server to another or between different types of database servers. In addition, the **onload** command must have the same scope as the corresponding **onunload** command that unloaded the same table or tables that **onload** references. You cannot, for example, use **onunload** to unload an entire database, and then use **onload** to load only a subset of the tables from that database.

The **onload** utility creates a database or table in a specified dbspace. The **onload** utility then loads it with data from an input tape or disk file that the **onunload** utility creates.

During the load, you can move simple large objects that are stored in a blobospace to another blobospace.

Syntax of the onunload command

The **onunload** command unloads data from a database and writes a database or table into a file on tape or disk.

onunload

-FILE option ¹⁵

Destination Parameters ¹⁶

database

:

owner.

table

-V

-version

Element	Purpose	Key Considerations
<i>database</i>	Specifies the name of a database	<p>Additional Information: The database name cannot be qualified by a database server name (<i>database@dbservername</i>).</p> <p>References: Syntax must conform to the Identifier segment; see Identifier on page .</p>
<i>owner.</i>	Specifies the owner of the table	<p>Additional Information: The owner name must not include invalid characters.</p> <p>References: For path name syntax, see your operating-system documentation.</p>
<i>table</i>	Specifies the name of the table	<p>Requirement: The table must exist.</p> <p>References: Syntax must conform to the Identifier segment; see Identifier on page .</p>

If you do not specify any destination parameter options, **onunload** uses the device that TAPEDEV specifies. The block size and tape size are the values specified as TAPEBLK and TAPESIZE, respectively. (For information about TAPEDEV, TAPEBLK, and TAPESIZE, see your *Informix® Administrator's Reference*.)

The **-V** option displays the software version number and the serial number. The **-version** option extends the **-V** option to display additional information about the build operating system, build number, and build date.

onunload destination parameters

The **onunload** utility supports tape or file destination options.

The following syntax diagram fragment shows **onunload** destination parameters

15. See [The -FILE option on page](#) .

16. See [onunload destination parameters on page 120](#)

Destination parameters

-l**-b** *blocksize***-s** *tapesize***-t** *source*

17

Element	Purpose	Key Considerations
-b <i>blocksize</i>	Specifies in kilobytes the block size of the tape device	Requirement: The <i>blocksize</i> must be an integer. Additional Information: This option overrides the default value in TAPEBLK or LTAPEBLK.
-l	Directs onunload to read the values for tape device, block size, and tape size from LTAPEDEV, LTAPEBLK, and LTAPESIZE, respectively	None.
-s <i>tapesize</i>	Specifies in kilobytes the amount of data that can be stored on the tape	Requirement: The <i>tapesize</i> must be an integer. To write to the end of the tape, specify a tape size of 0. If you do not specify 0, then the maximum <i>tapesize</i> is 2 097 151 KB. Additional Information: This option overrides the default value in TAPESIZE or LTAPESIZE.
-t <i>source</i>	Specifies the path name of the file on disk or of the tape device where the input tape is mounted	Additional Information: This option overrides the tape device specified by TAPEDEV or LTAPEDEV. The path name must be a valid path name.

Constraints that affect onunload

When you use the **onunload** utility, you must be aware of constraints that affect how you load the data on the **onunload** tape.

The following constraints apply to **onunload**:

- You must load the data on the **onunload** tape into a database or table that your database server manages.
- You cannot use **onunload** and **onload** if the databases contain extended data types.
- You must load the tape that **onunload** writes onto a computer with the same page size and the same representation of numeric data as the original computer.
- You must read the file that **onunload** creates with the **onload** utility of the same version of your database server. You cannot use **onunload** and **onload** to move data from one version to another.

17. Only one occurrence of each option allowed. More than one option can occur in a single invocation.

- When you unload a complete database, you cannot modify the ownership of database objects (such as tables, indexes, and views) until after you finish reloading the database.
- When you unload and load a table, **onunload** does not preserve access privileges, synonyms, views, constraints, triggers, or default values that were associated with the original tables. Before you run **onunload**, use the **dbschema** utility to obtain a listing of the access privileges, synonyms, views, constraints, triggers, and default values. After you finish loading the table, use **dbschema** to re-create the specific information for the table.

Privileges for database or table unloading

To unload a database, you must have DBA privileges for the database or be user **informix**. To unload a table, you must either own the table, have DBA privileges for the database in which the table resides, or be user **informix**.

User **root** does not have special privileges with respect to **onunload** and **onload**.

Tables that are unloaded with a database

If you unload a database, all of the tables in the database, including the system catalog tables, are unloaded.

All triggers, SPL routines, defaults, constraints, and synonyms for all of the tables in the database are also unloaded.

Data that is unloaded with a table

If you unload a table, **onunload** unloads the table data and information from the **systables**, **syscolumns**, **sysindexes**, and **sysblobs** system catalog tables.

When you unload a table, **onunload** does not unload information about constraints, triggers, or default values that are associated with a table. In addition, access privileges that are defined for the table and synonyms or views that are associated with the table are not unloaded.

Locking during unload operation

During the unload operation, the database or table is locked in shared mode. An error is returned if **onunload** cannot obtain a shared lock.

The **onload** utility creates a database or table in a specified dbspace. The **onload** utility then loads it with data from an input tape or disk file that the **onunload** utility creates.

Logging mode

The **onunload** utility does not preserve the logging mode of a database. After you load the database with **onload**, you can make a database ANSI compliant or add logging.

For information about logging modes, refer to the *Informix® Guide to SQL: Syntax*.

During the load, you can move simple large objects that are stored in a blob space to another blob space.

If you do not specify any source-parameter options, **onload** uses the device that is specified as TAPEDEV. The block size and tape size are the values that are specified as TAPEBLK and TAPESIZE, respectively. (For more information about TAPEDEV, TAPEBLK, and TAPESIZE, refer to your *Informix® Administrator's Guide*.)

If you do not specify creation options, **onload** stores the database or table in the root db space.

Syntax of the onload command

The **onload** command loads data that was created with the **onunload** command into the database server.

onload

-FILE option¹⁸

Source Parameters¹⁹

-ddbspacedatabase

:

owner.

table

Create Options²⁰

-V

-version

Element	Purpose	Key Considerations
-d <i>dbspace</i>	Loads a database or table into the specified db space	The tape being loaded must contain the specified database or table.
<i>database</i>	Specifies the name of the database	The database name cannot include a database server name, such as <i>database@dbservername</i> . References: Syntax must conform to the Identifier segment; see Identifier on page .
<i>owner.</i>	Specifies the owner of the table	The owner name must not include invalid characters. References: For path name syntax, refer to your operating-system documentation.
<i>table</i>	Specifies the name of the table	The table must exist. References: Syntax must conform to the Identifier segment; see Identifier on page .

18. See [The -FILE option on page](#).

19. See [onload source parameters on page 124](#)

20. See [onload create options on page 124](#)

The **-V** option displays the software version number and the serial number. The **-version** option extends the **-V** option to display additional information about the build operating system, build number, and build date.

onload source parameters

The **onload** command includes options for specifying information about the tape or file source.

The following syntax diagram fragment shows **onload** source parameters.

Source parameters

-l

-b *blocksize*

-s *tapesize*

-t *source*

21

Element	Purpose	Key Considerations
-b <i>blocksize</i>	Specifies in kilobytes the block size of the tape device	Requirements: Unsigned integer. Must specify the block size of the tape device. Additional Information: This option overrides the default value in TAPEBLK or LTAPEBLK.
-l	Directs onload to read the values for tape device, block size, and tape size from the configuration parameters LTAPEDEV, LTAPEBLK, and LTAPESIZE, respectively	Additional Information: If you specify -l , and then -b , -s , or -t , the value that you specify overrides the value in the configuration file.
-s <i>tapesize</i>	Specifies in kilobytes the amount of data that the database server can store on the tape	Requirements: Unsigned integer. To write to the end of the tape, specify a tape size of 0. If you do not specify 0, then the maximum <i>tapesize</i> is 2 097 151 KB. Additional Information: This option overrides the default value in TAPESIZE or LTAPESIZE.
-t <i>source</i>	Specifies the path name of the file on disk or of the tape device where the input tape is mounted	Must be a legitimate path name. Additional Information: This option overrides the tape device that TAPEDEV or LTAPEDEV specifies. References: For path name syntax, see your operating-system documentation.

21. Only one occurrence of each option allowed. More than one option can occur in a single invocation.

onload create options

The **onload** command includes information that is used to recreate the database.

The following syntax diagram fragment shows **onload** create options.

Create options

-coldcnstrntnewcnstrnt

-ioldindexnewindex

-fdolddbspnewdbsp

-fiindexnameolddbspnewdbsp

database

:

owner.

table

Element	Purpose	Key Considerations
-c <i>oldcnstrnt</i> <i>newcnstrnt</i>	Directs onload to rename the specified constraint.	None.
-i <i>oldindex</i> <i>newindex</i>	Directs onload to rename the table index when it stores the index on disk.	Additional Information: Use the -i option to rename indexes during the load to avoid conflict with existing index names. References: Syntax must conform to the Identifier segment; see Identifier on page .
-fd <i>olddbsp</i> <i>newdbsp</i>	Moves a data fragment from one dbspace to another.	The new dbspace must exist and must not already contain another data fragment for the table. Additional Information: This option is used with parallel data query (PDQ) and table fragmentation.
-fi <i>indexname</i> <i>olddb</i> <i>newdbsp</i>	Moves index fragments from one dbspace to another.	The new dbspace must exist and must not already contain another index fragment for the table. Additional Information: This option is used with PDQ and table fragmentation.
<i>database</i>	Specifies the name of the database	Requirement: The database name cannot include a database server name, such as <i>database@dbservername</i> . References: Syntax must conform to the Identifier segment; see Identifier on page .
<i>owner.</i>	Specifies the owner of the table	Requirement: The owner name must not include invalid characters. References: For path name syntax, refer to your operating-system documentation.

Element	Purpose	Key Considerations
<i>table</i>	Specifies the name of the table	<p>Requirement: The table must not exist.</p> <p>References: Syntax must conform to the Identifier segment; see Identifier on page .</p>

If you do not specify any create options for non-fragmented tables, the **onload** utility stores the database or table in the root dbspace.

For fragmented tables, **onunload** preserves the fragmentation expression for later use by **onload**. Thus an imported table is fragmented in the same way as the original table.

You can use the **-c**, **-i**, **-fd**, and **-fi** options in any order and as often as necessary as long as you use unique pairs.

Constraints that affect onload

The **onload** utility performs faster than the **dbimport**, **dbload**, or **LOAD** methods. In exchange for this higher performance, **onload** has certain constraints.

The **onload** utility has the following constraints:

- The **onload** utility only creates a new database or table; you must drop or rename an existing database or table of the same name before you run **onload**. During execution, the **onload** utility's prompt will ask you if you want to rename blobspaces.
- The **onload** utility places a shared lock on each of the tables in the database during the load. Although you cannot update a table row with the lock in place, the database is available for queries.
- When you load a complete database, the user who runs **onload** becomes the owner of the database.
- The **onload** utility creates a database without logging; you must initiate logging after **onload** loads the database.
- When you use **onload** to load a table into a logged database, you must turn off logging for the database during the operation.
- For fragmented tables, the dbspace assignment is preserved, unless you override it using the **-fn** option.
- For non-fragmented tables, the **onload** utility attempts to store the table in root dbspace if a target dbspace is not specified with the **-d** option. If storing the table in root dbspace or in the dbspace specified with the **-d** option is not possible due to difference in page sizes, the **onload** utility tries to use a dbspace that has the same dbspace number as the dbspace number of the originally unloaded table. If this dbspace still has a different page size, the load operation will fail.

Logging during loading

When you use the **onload** utility to create tables from an **onunload** input tape, **onload** can load information only into a database without logging. Thus, before you load a table into an existing, logged database, you must end logging for the database.

You also might want to consider loading during off-peak hours. Otherwise, you might fill the logical-log files or consume excessive shared-memory resources. After you load the table, create a level-0 dbspace backup before you resume database logging.

When you use **onload** to create databases from an **onunload** input tape, the databases that result are not ANSI-compliant and do not use transaction logging. You can make a database ANSI compliant or add logging after you load the database.

The **onload** utility performs all its loading within a transaction. This feature allows the changes to be rolled back if an error occurs.

Movement of simple large objects to a blobspace

If you load a table that contains simple large objects stored in a blobspace, the **onload** utility asks you if you want to move them to another blobspace.

If you respond **yes**, **onload** displays the blobspace name where the simple large objects were stored when the tape was created. It then asks you to enter the name of the blobspace where you want the simple large objects stored.

If you enter a valid blobspace name, **onload** moves all simple-large-object columns in the table to the new blobspace. Otherwise, **onload** prompts you again for a valid blobspace name.

Ownership and privileges

When you load a new database, the user who runs the **onload** utility becomes the owner. Ownership within the database (tables, views, and indexes) remains the same as when the database was unloaded to tape with **onunload**.

To load a table, you must have the Resource privilege on the database. When **onload** loads a new table, the user who runs **onload** becomes the owner unless you specify an owner in the table name. (You need the DBA privilege for the database to specify an owner in the table name.)

The **onunload** utility does not preserve synonyms or access privileges. To obtain a listing of defined synonyms or access privileges, use the **dbschema** utility, which [The dbschema utility on page 98](#) describes, before you run **onunload**.

Exclusive locking during a load operation

During a load operation, the **onload** utility places an exclusive lock on the new database or table.

Loading proceeds as a single transaction, and **onload** drops the new database or table if an error or system failure occurs.

Moving a database between computers with the onunload and onload utilities

You can use the **onunload** and **onload** utilities to move a complete database from one computer to another.

To move a database from one computer to another:

1. Make sure that the page size, numeric representations, and byte alignment on structures and unions are the same on both computers.

The page size is 2 KB on certain UNIX™ systems and 4 KB on Windows™. For information about page size, see your *Informix® Administrator's Guide*. The numeric representation and the byte alignment are characteristics of your operating system. For information about numeric representation and byte alignment, refer to the manuals for your operating systems.

2. Decide where to store the unloaded data:

- On disk. Create an empty file for onunload to hold the data. Make sure that you have write permission for the file.
- On tape. Use the tape device and characteristics specified in the ONCONFIG configuration file by either the TAPEDEV or LTAPEDEV configuration parameter, or specify another tape device. Make sure that the tape device that you specify is available for onunload. However, if you set the TAPEDEV configuration parameter to STDIO, the onunload utility will not be able to unload data.

3. Run the oncheck utility to make sure that your database is consistent.

For information about oncheck, see your *Informix® Administrator's Reference*.

4. Run the onunload utility to unload the data from the database.

For details on the syntax of the onunload command, see [Syntax of the onunload command on page 119](#).

5. If necessary, transfer the storage medium (tape or disk) to the new computer.

If the two computers are on the same network, you can read or write the data remotely.

6. Run the onload utility to load the data into the new database.

For details on the syntax of the onload command, see [Syntax of the onload command on page 123](#).

7. Set the logging status for the new database.

For information about logging status, see your *Informix® Administrator's Guide*.

8. If necessary, change the DBA privileges of the database.

9. Create a level-0 backup of the new database.

Moving a table between computers with the onunload and onload utilities

You can use the **onunload** and **onload** utilities to move a table from one computer to another.

To move a table from one computer to another:

1. Make sure that the page size, numeric representations, and byte alignment on structures and unions are the same on both computers. (The page size is 2 KB on certain UNIX™ systems and 4 KB on Windows™.)
2. Decide where to store the unloaded data.
3. Run the **oncheck** utility to make sure that your database is consistent.
4. If you want to save the triggers, access privileges, SPL routines, defaults, constraints, and synonyms for the table, run the **dbschema** utility.
5. Run the **onunload** utility.

For details on the syntax of the **onunload** command, see [Syntax of the onunload command on page 119](#).

6. If necessary, transfer the storage medium to the new computer.
7. If the table includes simple large objects that are stored in blobspaces, decide where to store the simple large objects. If necessary, create new blobspaces.
8. Turn off logging.

When you are loading a table, logging on the target database must be turned off. (When you are creating and loading an entire database, the logging status does not matter.)

9. Run the **onload** utility.

For details on the syntax of the **onload** command, see [Syntax of the onload command on page 123](#).

10. Create a level-0 backup of the modified database.
11. Turn logging back on, if you want logging.
12. If you want to restore the triggers, access privileges, SPL routines, defaults, constraints that are not preserved, and synonyms for the table, run the **dbschema** utility or recreate these objects manually.

Constraints such as primary keys or default values are preserved, even for a single table. Foreign keys, access privileges, SPL routines and synonyms are not preserved.

Moving a table between dbspaces with the onunload and onload utilities

You can use the **onunload** and **onload** utilities to move a table from one dbspace to another dbspace on the same computer.

To move a table from one dbspace to another dbspace on the same computer:

1. Run the **onunload** utility to unload the table.

For details on the syntax of the **onunload** command, see [Syntax of the onunload command on page 119](#).

2. Turn off logging.

When you are loading a table, logging on the target database must be turned off.

3. Run the **onload** utility.

Specify a new table name and new dbspace name in the **onload** command.

For details on the syntax of the **onload** command, see [Syntax of the onload command on page 123](#).

4. If the data loads successfully, delete the old table in the old dbspace and rename the new table to the old table name.
5. Create a level-0 backup of the modified database.
6. Turn logging back on, if you want logging.

The onmode utility reversion option

You use the **-b** option of the **onmode** utility to revert to the older database server from which you converted.

What the onmode -b command does

When you convert a database server, several modifications make the format of the databases incompatible with the older version. The `onmode -b` command modifies data so that the earlier version of the database server can access it.

When you convert a database server, several modifications make the format of the databases incompatible with the older version. The `onmode -b` command modifies data so that the earlier version of the database server can access it. In some cases the format of the databases is compatible between versions and the `onmode -b` command is not needed. Type `onmode -b` to see the usage message for options that are available for your database server.

The utility does not revert changes made to the layout of the data that do not affect compatibility.

You must revert the databases before users can access the data with the earlier database server version.

For information about other **onmode** options, see [The onmode utility on page](#) in your *Informix® Administrator's Reference*.

Related information

[Run the reversion utility on page 58](#)

Preparation for using the onmode -b command

Before you use the `onmode -b` command, notify users that you are going to bring the database server offline. The reversion utility forcibly removes all users and shuts down the database server.

The **onmode -b** command includes an implicit **-yuk** command.

Make sure that the **INFORMIXSERVER** environment variable is set to the correct database server.

UNIX/Linux Only

You must be user **root** or user **informix** to run **onmode**.

Windows™ Only

You must be a member of the **Informix-Admin** group to run **onmode**.


Related information

[Run the reversion utility on page 58](#)

Syntax of the onmode -b command

The `onmode -b` command restores your databases to the version of Informix® from which you converted. You cannot use this command to revert to any other version of the server.

When you convert a database server, several modifications make the format of the databases incompatible with the older version. The `onmode -b` command modifies data so that the earlier version of the database server can access it. In some cases the format of the databases is compatible between versions and the `onmode -b` command is not needed.

 **Tip:** To see a list of all of the versions to which you can revert, run this command:

```
onmode -b
```

When you see the options that are available, choose the option that is closest to the version that you want.

Figure 4. Syntax

onmode

-b *version_number*

Element	Purpose
<code>-b version_number</code>	Reverts the database to the specified version.

Related information


[Run the reversion utility on page 58](#)

The onrestorept utility

You can use the `onrestorept` utility to restore the database server back to the state that it was in just before the start of the failed upgrade.

The utility depends on these configuration parameters being set correctly before you attempt to upgrade the database server:

- The `CONVERSION_GUARD` configuration parameter must be set to `1` or `2` (the default value) for data to be stored during an upgrade. If the conversion guard operations fail (for example, because the server has insufficient space to store the data that is captured during the upgrade), you cannot use the `onrestorept` utility.
- The `RESTORE_POINT_DIR` configuration parameter must specify a directory where the conversion guard utility can store data files during the upgrade. Those files are needed to restore the database server to a consistent state after a failed upgrade. This directory must be empty before the upgrade starts. After a successful upgrade, the contents of the directory are automatically removed.

 **Important:** Informix® must be offline when you run the `onrestorept` utility. Do not start the server until the utility finishes running. Executing the `onrestorept` utility when the server is online, or starting the server before the utility finishes running, can damage the database and requires you to restore the database server from a backup copy.

To start Enterprise Replication after the `onrestorept` utility restores the database server to a consistent state, you must use the `cdr cleanstart` command.

Related information

[Restoring to a previous consistent state after a failed upgrade on page 45](#)

[Preparing for migration on page 7](#)

Syntax of the onrestorept command

The onrestorept command undoes changes made during a failed upgrade, restoring files to the state they were in when you shut down the server.

onrestorept**-V****-c****-y**

Element	Purpose	Key Considerations
-V	Display the version of the current server and the software serial number.	
-c	After a failed upgrade, delete the files in the directory specified in the RESTORE_POINT_DIR configuration parameter.	Before you begin another upgrade, you must delete these restore point files. Do this before you make another migration attempt, but not before you run the onrestorept utility to recover the files (if possible). If the upgrade was successful, restore point files are automatically deleted and there is no need to run onrestorept -c.
-y	Automatically accept every prompt while the onrestorept command runs.	If you do not specify -y, you must respond to every prompt.

Example**Examples**

The following command restores Informix® files after a failed upgrade:

```
onrestorept
```

The following command removes restore point files after a failed upgrade:

```
onrestorept -c
```

Index

A

- Abbreviated years 65
- ALARMPROGRAM configuration parameter 42, 44
- ANSI joins 52

B

- Backups
 - after upgrading 50
 - before reverting 55
 - before upgrading to a new version 15
 - logical logs 44
 - ON-Bar utility 15, 61
 - ontape utility 15, 61
 - source database 15
- Binary files, loading 117, 118
- BladeManager
 - installing and registering DataBlade modules 50, 62
 - removing new extensions before reversion 57
- Blobspaces
 - moving, with onunload and onload 127, 128
- BSON date fields 12, 47, 60
 - migrating 12

C

- Character-position form of FILE and INSERT statements 91
- Checking available space
 - before migration 8
- Chunks
 - reverting reserve pages 53
- client applications 48
- client compatibility 48
- Clusters
 - Apply fix pack or PID 25
 - migrating to new release 24
 - migrating to new version 29
 - restoring from a backup archive 39
 - restoring from the HDR secondary server 40
 - reverting 24, 37
 - upgrading 31
 - upgrading to a new fix pack 28
 - upgrading to a new PID 28
 - upgrading with conversion 29
- Command file
 - dbload 86
- Communications Support Module
 - configuring after migration 44
 - removing if reverting 59
 - saving before reverting 54
- concsn.cfg file
 - creating entries after migration 44
 - removing if reverting 59
 - saving before reverting 54
- Configuration file
 - customizing after migration 42
 - replacing after reversion 59
 - saving before migration 11
 - saving before reverting 54
- Configuration parameters
 - ALARMPROGRAM 42, 44
 - CONVERSION_GUARD 45, 131
 - HPL_DYNAMIC_LIB_PATH 42
 - RESTORE_POINT_DIR 45, 131
 - ROOTOFFSET 42

- ROOTPATH 42
- ROOTSIZE 42
- STOP_APPLY 65
- UPDATABLE_SECONDARY 65
- USELASTCOMMITTED 65
- CONVERSION_GUARD configuration parameter 45, 131
- convTovNoSQL1210.sql script 47, 60

D

- Data integrity 49
- data migration
 - external tables 64
- Data migration
 - constraints 2
 - issues to consider 1
 - overview 1
 - prerequisites 2
 - tools 2
- data types
 - SQLINTEGER 48
 - SQLLEN 48
 - SQLUINTEGER 48
 - SQLULEN 48
- Database server
 - initializing after upgrading 44
 - new version
 - performance tuning 50
 - reverting 51
 - reverting from current version 57
 - starting after upgrading 44
- Database servers
 - migrating 40, 40, 46
 - preparing for migration 6
 - upgrading 40, 40, 46
- Databases
 - ownership, set by onload 127
- DataBlade modules
 - installing after upgrading 44
 - registering 50, 62
- DB-Access
 - input from the dbschema utility 114, 114
- dbexport
 - SELECT triggers, disabling 65
- dbexport utility 65
 - c option 67, 70
 - d option 67
 - nw option 67
 - q option 67
 - si option 67, 70
 - ss option 67, 70
 - V option 67
 - version option 67
 - X option 67
 - destination options 71, 71
 - Interrupt key 70
 - schema output 73
 - syntax 67
- dbimport utility 65
 - c option 74, 76
 - D option 74
 - l option 80
 - nv option 74
 - q option 74
 - V option 74
 - version option 74
 - X option 74
 - create options 78
- database logging mode 80
- importing from another computer 5
- input file location options 76, 76
- Interrupt key 76
- locale, changing 81
- renaming a database 80
- syntax 74
- using with GLS 74
- using with NLS 81

- dbload utility
 - c command file option 82
 - d database option 82
 - e errors option 82
 - e option 85
 - i ignore rows option 82
 - i option 85
 - k option 82
 - l error log file option 82
 - p option 82
 - r option 82, 84
 - s option 82
 - V option 82
 - version option 82
 - X option 82
 - compared to LOAD 82
 - creating a command file 86
 - dbload utility
 - n commit interval option 82
 - FILE statement 86
 - guidelines for handling objects 85
 - ignoring rows 85
 - importing from another computer 5
 - INSERT statements 86
 - compared to SQL INSERT statement 91
 - using 87
 - Interrupt key 85
 - number errors to allow 85
 - overview 82
 - speed, increasing 85
 - syntax 82
 - table locking 84
 - writing a command file
 - in character-position form 93
 - in delimiter form 89
- dbschema utility
 - ss option 104
 - u option 104
 - chunk schema 107, 107
 - create schema across a network 103
 - create schema for a database 103
 - distribution information 111
 - example of file for DB-Access 114
 - guidelines 99
 - log schema 107, 107
 - output example 112
 - overview 98
 - owner conventions 103
 - privileges information 110, 111
 - privileges information for a role 111
 - re-creating the schema 114
 - sequence schema 105
 - specifying a table, view, or procedure 106
 - storage space schema 107, 107
 - synonym schema 106
 - syntax 99
 - syntax for role schema 109
- DBSECADM role 65
- dbspaces

- moving tables to another dbspace 129
- Delimiter form of FILE and INSERT statements 87, 87, 89
- Diagnostic information to gather 15
- Directories
 - installation 41
- Distributed queries
 - with ANSI joins 52

E

- Environment variables
 - DB_LOCALE 81
 - DBCENTURY 65
 - DBTEMP 81
 - GL_DATE 65
 - GL_DATETIME 117
 - GL_USEGLU 42
 - INFORMIXSERVER 42
 - INFORMIXSQLHOSTS 42
 - ONCONFIG 42
 - PATH 42
 - resetting after reversion 59
 - TEMP 81
 - TMP 81
 - USE_DTENV 117
- Exporting
 - time series data 72
- external tables 64
- Extracting schema information 62

F

- Fast recovery
 - initiating 13
- Features
 - reviewing 8
- FILE statement
 - character-position form 91
 - delimiter form 87, 87, 89
 - syntax for
 - character-position form 91
 - delimiter form 87
 - with dbload 86
- FIRST clause 52
- Fix packs
 - Apply to clusters 25

G

- GL_DATETIME environment variable 65, 117
- GL_USEGLU environment variable 42
- Global Language Support (GLS)
 - dbimport utility 74
 - using onload and onunload 118
- GRANT statement
 - role privileges 111

H

- HEX binary data 74
- HPL_DYNAMIC_LIB_PATH configuration parameter 42

I

- id_column 55
- Importing
 - non-
 - Informix
 - data
 - 5
- in dbschema output 111, 111
- In-place alters
 - oncheck -pT command 55
 - Sample updates 55
- Index

- rebuilding 44
- Informix
 - installing 41
- INFORMIXDIR directory 41
- INFORMIXSERVER environment variable 42
- INFORMIXSQLHOSTS environment variable 42
- Initializing
 - after upgrading 44
- INSERT statements
 - character-position form 91
 - delimiter form 87
 - syntax for character-position form 91
 - with dbload 86
- Installation directory 41
- Installing
 - Informix
 - 41

J

- Java UDRs 60
- JSON compatibility 47, 60

L

- Label-based access control (LBAC) 65
- LATERAL keyword 52
- Level-0 backup 15, 50, 61
 - after moving data 127
- LIMIT keyword 52
- LOAD SQL statement
 - for locales that support multibyte code sets 116
 - overview 115
 - syntax 116
- LOAD statement
 - for non-default GL_DATETIME environment variable settings 117
 - for non-default locales 117
- Loading
 - ASCII files 5
 - binary data 117, 118
 - data 2, 5
- Loading
 - set by onload 127
- Logical log
 - backup 44
 - out of space 44
 - space required for migration 8
- LTAPEDEV configuration parameter
 - onunload/onload 127

M

- Migrating
 - between 32-bit and 64-bit database servers 16
- Migrating a database
 - constraints 2
 - issues to consider 1
 - overview 1
 - planning for 2
 - prerequisites 2
 - to a new operating system 62, 62
 - tools 2
- Migrating with Enterprise Replication 17, 18
- Migration
 - before you begin 6
 - checklist 15
 - diagnostic information that you need before upgrading 15
 - monitoring status with online.log 42
 - onload utility 119
 - onunload utility 119
 - planning 7

- preparing for 7
- prerequisites 7
- space requirements 8
- with
 - Enterprise Replication
 - 17
 - with HDR, RS, and SD servers 21, 24
 - with high-availability clusters 21, 24, 28, 29, 31
 - with secondary servers 29

Mode

- checking 14
- modifying
 - client applications 48
- Monitoring migration status 42
- Moving data
 - blobspaces 128
 - constraints 2
 - overview 1
 - using dbexport and dbimport 65
 - using dbload 82
 - using distributed SQL 5
 - using onload and onunload 117, 117
 - using onunload and onload 127, 128, 129
 - when changing operating systems 62, 62, 63, 64
- Multi-node Active Clusters for High Availability (MACH)
 - Clusters
 - Migrating to new release 21

N

- Native Language Support (NLS)
 - populating with dbimport 81
- Non-
 - Informix
 - data, importing
 - 5
- NOVALIDATE constraint mode 74

O

- ON-Bar utility 47
 - backing up
 - after upgrading 50
 - backing up before upgrading 15
- oncheck utility
 - cc database_name option 14
 - cD database_name option 14
 - ce option 14
 - cl database_name option 14
 - cr option 14
 - rebuilding table indexes 44
 - verifying database integrity 14, 49
- ONCONFIG environment variable 42
- ONCONFIG file
 - customizing after migration 42
- oninit utility
 - s option 13
- online.log 42
- onload and onunload utilities 127, 128, 129
- onload utility
 - constraints 126
 - constraints on use 118
 - create options 124
 - handling large objects in a blobspace 127
 - how it works 119
 - logging status 126
 - moving a database 127
 - moving a table 128, 129, 129
 - moving locales 118
 - moving to another dbspace 129
 - ownership and privileges 127

- specifying source parameters 124
- syntax 123
- using between computers 117, 117
- onmode -b command 129, 129
- onmode utility
 - b option 58
 - ky option 13
 - sy option 13
 - reverting from the current version 58
 - shutting down 13
 - shutting down the server 13
- onmode-b command 130
 - syntax 130
- onrestorept utility
 - Clusters
 - restoring primary server to a consistent point 39
 - overview 131
 - syntax 132
 - undoing failed upgrade changes 39, 45
- onstat utility 14
- ontape utility
 - a option 44
 - backing up
 - after upgrading 50
 - before upgrading 15
- onunload utility
 - constraints on use 118, 121
 - destination parameters 120
 - how it works 119
 - locking 122
 - logging mode 122
 - moving a database 127
 - moving a table 128, 129, 129
 - moving locales 118
 - moving to another dspace 129
 - ownership and privileges 122
 - syntax 119
 - unloading tables 122
 - using between computers 117, 117
 - what is included with a
 - database 122
 - table 122
- Operating system
 - adjusting tables after changing operating systems 63
 - moving data to another one 62, 62, 63, 64
- ORDER BY clause 52

P

- PATH environment variable 42
- Performance tuning
 - adjusting queries after upgrading 50
 - after upgrading 50
- PID
 - Apply to clusters 25
- Planning
 - before moving data 2
 - data migration 1
 - for exporting and importing data 62
 - for migration 7
 - for upgrading your server 7
- Platforms, moving data between
 - compatible computers 5
- post-migration steps 47, 60
- Privileges 111, 111
 - required for onunload 121

Q

- Queries
 - adjusting after upgrading 50
- Quiescent mode 14

R

- recompiling
 - client applications 48
- Recompiling Java UDRs 60
- Registering DataBlade modules 50, 62
- restore points 131
- RESTORE_POINT_DIR configuration parameter 45, 131
- Reverse migration 51
- Reversion utility 58
- Revert to original version 51
- Reverting
 - backing up before you start 55
 - before using the onmode -b command 130
 - chunk reserve pages 53
 - database schema 52
 - from the new version 51
 - from
 - Version 14.10
 - 57
 - limitations 52
 - remove features 57
 - restrictions for 52
 - restrictions for reverting to prior versions 52
 - using the onmode -b command 129, 129
 - with Enterprise Replication 20
 - with HDR, RS, and SD servers 24, 37
 - with high-availability clusters 24, 37
- Rolling upgrade
 - with temporary Enterprise Replication 31
- Rolling upgrades 25
- ROOTOFFSET configuration parameter 42
- ROOTPATH configuration parameter 42
- ROOTSIZE configuration parameter 42
- Running the reversion utility 58

S

- Sample updates 55
- Schema
 - create across a network 103
 - create for a database 103
 - display with dbschema 98
- scripts
 - convTovNoSQL1210.sql 47, 60
- Scripts
 - concdr.bat 18
 - concdr.sh 18
- SELECT triggers, disabling with dbexport 65
- Simple large objects
 - moving with onload 127, 127, 128, 128
- SKIP keyword 52
- Slow queries
 - adjusting after upgrading 50
- sm_versions file 47
- smi_unld utility 54
- Space
 - checking availability before migration 8
 - for sysmaster database 8
- SQL statements
 - UPDATE STATISTICS
 - data distributions 111
- sqlhosts file
 - save a copy when migrating 11
- sqlhosts file, UNIX
 - changing name or path 42
 - csm option 44
- Starting
 - after upgrading 44
 - server after reversion 60
- STOP_APPLY configuration parameter 65

- sysmaster database
 - and logical logs 44
 - space required for migration 8

T

- TAPEDEV configuration parameter, with onunload and onload 127
- Time series data
 - exporting 72
- Transactions
 - checking for open ones 13
- Truncate keyword 52

U

- UNLOAD SQL statement
 - for locales that support multibyte code sets 116
 - overview 115
 - syntax 115
- UNLOAD statement
 - for non-default GL_DATETIME environment variable settings 117
 - for non-default locales 117
- UPDATABLE_SECONDARY configuration parameter 65
- UPDATE statements
 - sample 55
- UPDATE STATISTICS statement 48
 - data distributions 111
 - using after reversion 61
- Upgrading your server
 - overview of tasks 40
 - planning 7
 - preparing for 7
 - preparing to undo changes 10
 - prerequisites 7
 - restoring files after a failure 45, 131
- USE_DTENV environment variable 65, 117
- USELASTCOMMITTED configuration parameter 65
- USELASTCOMMITTED session environment variable 65
- Using SSL/TLS database connections 43
- Utilities
 - dbexport 62, 65, 65
 - dbexport syntax 67
 - dbimport 62, 65
 - dbimport syntax 74
 - dbload 82
 - dbschema 98
 - onload 62, 123
 - onload and onunload 117, 127, 128, 129
 - onunload 62, 119

V

- Verifying data integrity 49