IBM DB2 Universal Database

Data Movement Utilities Guide and Reference

Version 6

SC09-2858-00

IBM DB2 Universal Database

Data Movement Utilities Guide and Reference

Version 6

SC09-2858-00

Before using this information and the product it supports, be sure to read the general information under “Appendix F. Notices” on page 247.

Order publications through your IBM representative or the IBM branch office serving your locality or by calling 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Contents

About This Book

This book provides information about, and shows you how to use, the following IBM DB2 Universal Database (UDB) data movement utilities:

- The Import and Export utilities move data between a table or view and another database or spreadsheet program; between DB2 databases; and between DB2 databases and host databases using DB2 Connect. The export utility moves data from a database into operating system files; you can then use those files to import or load that data into another database.
- The Load utility moves data into tables, extends existing indexes, and generates statistics. Load moves data much faster than the import utility when large amounts of data are involved. Data unloaded using the export utility can be loaded with the load utility.
- The AutoLoader utility splits large amounts of data and loads the split data into the different partitions of a partitioned database.
- DataPropagator (DPROP) is a component of DB2 Universal Database that allows automatic copying of table updates to other tables in other DB2 relational databases.

Other vendor’s products that move data in and out of databases are also available, but are not discussed in this book.

Who Should Use this Book

This manual is for database administrators, application programmers, and other DB2 UDB users who perform the following tasks:

- Load data into DB2 tables from operating system files
- Move data between DB2 databases, and between DB2 and other applications (for example, spreadsheets)
- Archive data.

It is assumed that you are familiar with DB2 Universal Database, Structured Query Language (SQL), and with the operating system environment in which DB2 UDB is running. For general information about DB2 UDB, see the *Administration Guide*. For information about SQL, see the *SQL Reference*. For information about configuring, invoking, and using the DB2 UDB command line processor, see the *Command Reference*. For information about the DB2 UDB application programming interfaces (APIs), see the *Administrative API Reference*. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*. This manual does not contain instructions for installing DB2, which depend on your operating system. Installation information can be found in the appropriate *Quick Beginnings* book for your operating system.

How this Book is Structured

The following topics are covered:

Chapter 1
Describes the DB2 export utility, used to move data from DB2 tables into files.

Chapter 2
Describes the DB2 import utility, used to move data from files into DB2 tables or views.

Chapter 3
Describes the DB2 load utility, used to move large volumes of data from files into DB2 tables.

Chapter 4
Describes the AutoLoader utility, which splits large amounts of data and loads the split data into the different partitions of a partitioned database.

Chapter 5
Describes how to use the DB2 export, import, and load utilities to move DB2 File Manager data.

Chapter 6
Describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from DRDA host databases. DataPropagator (DPROP), another method for moving data between databases in an enterprise, is also described.

Appendix A.
Explains the conventions used in syntax diagrams.

Appendix B
Summarizes the important differences between the DB2 load and import utilities.

Appendix C
Describes external file formats supported by the database manager export, import, and load utilities.

Appendix D
Provides information about interpreting messages generated by the database manager when a warning or error condition has been detected.

Chapter 1. Export

This chapter describes the DB2 UDB export utility, which is used to write data from a DB2 database to one or more files stored outside of the database. The exported data can then be imported or loaded into another DB2 database, using the DB2 import or the DB2 load utility, respectively, or it can be imported into another application (for example, a spreadsheet).

The following topics are covered:

For information about exporting DB2 Data Links Manager data, see “Using Export to Move DB2 Data Links Manager Data” on page 149. For information about exporting data out of typed tables, see “Moving Data Between Typed Tables” on page 165. For information about exporting data from a DRDA server database to a file on the DB2 Connect workstation, and the reverse, see “Moving Data With DB2 Connect” on page 163.

Export Overview

The export utility exports data from a database to an operating system file, which can be in one of several external file formats.

The following information is required when exporting data:

- An SQL SELECT statement specifying the data to be exported.
- The path and name of the operating system file that will store the exported data.
- The format of the data in the input file. This format can be IXF, WSF, or DEL. See “Appendix C. Export/Import/Load Utility File Formats” on page 179.
- A message file name.
- When exporting typed tables, you may need to provide the subtable traverse order within the hierarchy. If the IXF format is to be used, the default order is recommended. When specifying the order, recall that the subtables must be traversed in the PRE-ORDER fashion. When exporting typed tables, you cannot provide a SELECT statement directly. Instead, you must specify the target subtable name, and optionally a WHERE clause. The export utility uses this information, along with the traverse order, to generate and execute the required SELECT statement. For more information, see “Moving Data Between Typed Tables” on page 165.

You can also specify:

- New column names when exporting to IXF or WSF files. If you do not want to specify new column names, the column names in the existing table or view are used in the exported file.
- Additional options to customize the export operation (see “File Type Modifiers (Export)” on page 15).

If you want to use the export utility in a multiple database partition environment, you can use **db2batch** to complete the task at each database partition. The SELECT statement must be able to return only the data found locally. The selection condition is as follows:

```
SELECT * FROM tablename WHERE NODENUMBER(column-name) = CURRENT NODE
```

For more information about **db2batch**, see the *Command Reference* or the *Administration Guide*.

Privileges, Authorities, and Authorization Required to Use Export

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM or DBADM authority, or CONTROL or SELECT privilege for each table participating in the export operation.

Using Export

Before Using Export

Before invoking the export utility, you must be connected to (or be able to implicitly connect to) the database from which the data will be exported. Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking export. Other user applications accessing the table using separate connections need not disconnect.

Invoking Export

The export utility can be invoked through:

- The command line processor (CLP).

  Following is an example of the EXPORT command issued through the CLP:

  ```
  db2 export to staff.ixf of ixf select * from userid.staff
  ```

- The Export notebook in the Control Center. To open the Export notebook:

  1. From the Control Center, expand the object tree until you find the Tables or Views folder.
  2. Click on the folder you want to work with. Any existing tables or views are displayed in the pane on the right side of the window (the contents pane).
  3. Click the right mouse button on the table or view you want in the contents pane, and select Export from the pop-up menu. The Export notebook opens.

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.

- An application programming interface (API), **sqluexpr**. For information about this API, see “Export API” on page 8. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

Recreating an Exported Table

A table can be saved by using the export utility and specifying the IXF file format. The saved table (including its indexes) can then be recreated using the import utility. The export utility will fail if the data you want to export exceeds the space available on the file system on which the exported file will be created. In this case, you should limit the amount of data selected by specifying conditions on the WHERE clause, so that the export file will fit on the target file system. You can invoke the export utility multiple times to export all of the data.

Exporting Large Objects (LOBs)

When exporting data from large object (LOB) columns, the default action is to select the first 32KB of data, and to place this data in the same file as the rest of the column data. If the `lobsinfile` modifier (see “File Type Modifiers (Export)” on page 15) has been specified, the export utility selects the entire LOB (up to 2GB), and places it in a separate file.

EXPORT Command

Command Syntax

```
►►──EXPORT TO──filename──OF──filetype──┬─────────────────────────────┬──────────►
                                        │           ┌─,──────────┐    │
                                        └─LOBS TO───▼───lob-path─┴────┘

►──┬──────────────────────────────┬─┬──────────────────────────────────────┬───►
   │           ┌─,────────────┐   │ │                ┌──────────────────┐   │
   └─LOBFILE───▼───filename───┴───┘ └─MODIFIED BY────▼───filetype-mod───┴───┘

►──┬─────────────────────────────────────────┬─┬──────────────────────────┬────►
   │                ┌─,───────────────┐       │ └─MESSAGES──message-file───┘
   └─METHOD N──(────▼───column-name───┴───)───┘

►──┬─select-statement─────────────────────────────────────────────────┬─────────►◄
   └─HIERARCHY───┬─STARTING──sub-table-name───┬──┬──────────────────┬──┘
                 └─┤ traversal-order-list ├───┘  └─where-clause─────┘
```

traversal-order-list:

```
         ┌─,─────────────────┐
├──(─────▼───sub-table-name──┴────)──────────────────────────────────────────────┤
```

Command Parameters

HIERARCHY traversal-order-list

Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING sub-table-name

Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE filename

Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/path/bar.002`, and so on.

LOBS TO lob-path

Specifies one or more paths to directories in which the LOB files are to be stored. When file space is exhausted on the first path, the second path will be used, and so on.

MESSAGES message-file

Specifies the destination for warning and error messages that occur during an export operation. If the file already exists, the export utility appends the information. If *message-file* is omitted, the messages are written to standard output.

METHOD N column-name

Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY filetype-mod

Specifies additional options (see Table 2 on page 15).

OF filetype

Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony

  Note: When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values that do not fall within this range are also exported, importing or loading these values back may result in incorrect data, depending on the operating system.
- IXF (integrated exchange format, PC version), in which most of the table attributes, as well as any existing indexes, are saved in the IXF file, except when columns are specified in the SELECT statement.

With this format, the table can be recreated, while with the other file formats, the table must already exist before data can be imported into it.

For more information about file formats, see “Appendix C. Export/Import/Load Utility File Formats” on page 179.

select-statement

Specifies the SELECT statement that will return the data to be exported. If the SELECT statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO filename

Specifies the name of the file to which data is to be exported. If the complete path to the file is not specified, the export utility uses the current directory and the default drive as the destination.

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

Export API

C API Syntax

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqluexpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Export */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgexpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    sqlu_media_list * pLobFileList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluexpt_out*  pOutputInfo,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

DataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

FileTypeLen

Input. A 2-byte unsigned integer representing the length in bytes of the file type.

MsgFileNameLen

Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

pDataFileName

Input. A string containing the path and the name of the external file into which the data is to be exported.

pLobPathList

Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files are to be stored.

When file space is exhausted on the first path in this list, the API will use the second path, and so on.

For more information, see “SQLU-MEDIA-LIST ” in the *Administrative API Reference*.

pLobFileList

Input. An *sqlu_media_list* using *media_type* `SQLU_CLIENT_LOCATION`, and the *sqlu_location_entry* structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on.

For more information, see “SQLU-MEDIA-LIST ” in the *Administrative API Reference*.

When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *pLobFilePath*), and then appending a 3-digit sequence number. For example, if the current LOB path is the directory `/u/foo/lob/path`, and the current LOB file name is `bar`, the created LOB files will be `/u/foo/lob/path/bar.001`, `/u/foo/lob/pah/bar.002`, and so on.

pDataDescriptor

Input. Pointer to an *sqldcol* structure specifying the column names for the output file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by the export utility. Valid values for this parameter (defined in `sqlutil`) are:

SQL_METH_N

Names. Specify column names to be used in the output file.

SQL_METH_D

Default. Existing column names from the table are to be used in the output file. In this case, the number of columns and the column specification array are both ignored. The column names are derived from the output of the SELECT statement specified in *pActionString*.

For more information, see “SQLDCOL” in the *Administrative API Reference*.

pActionString

Input. Pointer to an *sqlchar* structure containing a valid dynamic SQL SELECT statement. The structure contains a 2-byte long field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from *pDataDescriptor*), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

For more information, see “SQLCHAR” in the *Administrative API Reference*.

Note: The syntax that is to be used for typed tables is described in “EXPORT Command” on page 5.

pFileType

Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

SQL_DEL

Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_WSF

Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

SQL_IXF

PC version of the Integrated Exchange Format, the preferred method for exporting data from a table. Data exported to this file format can later be imported or loaded into the same table or into another database manager table.

pFileTypeMod

Input. A pointer to an *sqldcol* structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see "SQLCHAR" in the *Administrative API Reference*, and "File Type Modifiers (Export)" on page 15.

pMsgFileName

Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is overwritten. If it does not exist, a file is created.

CallerAction

Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL

Initial call. This value must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested export operation, the caller action must be set to one of the following:

SQLU_CONTINUE

Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE

Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

pOutputInfo
Ouput. Returns the number of records exported to the target file. For more information about this structure, see “SQLUEXPT-OUT Data Structure” on page 14.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see “SQLCA” in the *Administrative API Reference*.

REXX API Syntax

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]

CONTINUE EXPORT

STOP EXPORT
```

REXX API Parameters

stmt A REXX host variable containing a valid dynamic SQL SELECT statement. The statement specifies the data to be extracted from the database.

datafile
Name of the file into which the data is to be exported.

filetype
The format of the data in the export file. The supported file formats are:

DEL Delimited ASCII

WSF Worksheet format

IXF PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options (see “File Type Modifiers (Export)” on page 15).

dcoldata
A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns (number of elements in the remainder of the variable).

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

If this parameter is NULL, or a value for *dcoldata* has not been specified, the utility uses the column names from the database table.

msgfile
File, path, or device name where error and warning messages are to be sent.

number
A host variable that will contain the number of exported rows.

SQLUEXPT-OUT Data Structure

This structure is used to pass information from the "Export API" on page 8.

Table 1. Fields in the SQLUEXPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of the structure.
ROWSEXPORTED	INTEGER	Number of records exported from the database into the target file.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQL-UEXPT-OUT */
/* ... */
SQL_STRUCTURE sqluexpt_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsExported;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UEXPT-OUT.
    05 SQL-SIZE-OF-UEXPT-OUT  PIC 9(9) COMP-5 VALUE 8.
    05 SQL-ROWSEXPORTED       PIC 9(9) COMP-5 VALUE 0.
*
```

File Type Modifiers (Export)

Table 2. Valid File Type Modifiers (Export)

Modifier	Description
All File Formats	
lobsinfile	*lob-path* specifies the path to the files containing LOB values.
DEL (Delimited ASCII) File Format	
chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[a] The single quotation mark (') can also be specified as a character string delimiter as follows: `modified by chardel''`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[a]
	In the following example, `coldel;` causes the export utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 "export to temp of del modified by coldel;`
	`   select * from staff where dept = 20"`

datesiso	Date format. Causes all date data values to be exported in ISO format.
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[a]
dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [ab] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
nodoubledel	Suppresses recognition of double character delimiters.
WSF File Format	

Table 2. Valid File Type Modifiers (Export) (continued)

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[b] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[b]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[b]
4	Creates a WSF file containing DBCS characters.

Notes:

1. The export utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the export operation fails, and an error code is returned.
2. [a] "Delimiter Restrictions" lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] These files can also be directed to a specific product by specifying an L for Lotus 1-2-3, or an S for Symphony in the *filetype-mod* parameter string. Only one value or product designator may be specified.

Delimiter Restrictions

It is the user's responsibility to ensure that the chosen delimiter character is not part of the data to be moved. If it is, unexpected errors may occur. The following restrictions apply to column, string, DATALINK, and decimal point delimiters when moving data:

- Delimiters are mutually exclusive.
- A delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space.
- The default decimal point (.) cannot be a string delimiter.
- The following characters are specified differently by an ASCII-family code page and an EBCDIC-family code page:
  - The Shift-In (0x0F) and the Shift-Out (0x0E) character cannot be delimiters for an EBCDIC MBCS data file.
  - Delimiters for MBCS, EUC, or DBCS code pages cannot be greater than 0x40, except the default decimal point for EBCDIC MBCS data, which is 0x4b.
  - Default delimiters for data files in ASCII code pages or EBCDIC MBCS code pages are:

    ```
    " (0x22, double quotation mark; string delimiter)
    , (0x2c, comma; column delimiter)
    ```
  - Default delimiters for data files in EBCDIC SBCS code pages are:

```
" (0x7F, double quotation mark; string delimiter)
, (0x6B, comma; column delimiter)
```

- The default decimal point for ASCII data files is 0x2e (period).
- The default decimal point for EBCDIC data files is 0x4B (period).
- If the code page of the server is different from the code page of the client, it is recommended that the hex representation of non-default delimiters be specified. For example,

  ```
  db2 load from ... modified by chardel0x0C coldelX1e ...
  ```

The following information about support for double character delimiter recognition in DEL files applies to the export, import, and load utilities:

- Character delimiters are permitted within the character-based fields of a DEL file. This applies to fields of type CHAR, VARCHAR, LONG VARCHAR, or CLOB (except when `lobsinfile` is specified). Any pair of character delimiters found between the enclosing character delimiters is imported or loaded into the database. For example,

  ```
  "What a ""nice"" day!"
  ```

  will be imported as:

  ```
  What a "nice" day!
  ```

  In the case of export, the rule applies in reverse. For example,

  ```
  I am 6" tall.
  ```

  will be exported to a DEL file as:

  ```
  "I am 6"" tall."
  ```

- In a DBCS environment, the pipe (|) character delimiter is not supported.

Example Export Sessions

CLP Examples

The following example shows how to export information from the STAFF table in the SAMPLE database (to which the user must be connected) to `myfile.ixf`, with the output in IXF format. If the database connection is not through DB2 Connect, the index definitions (if any) will be stored in the output file; otherwise, only the data will be stored:

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database (to which the user must be connected) to `awards.ixf`, with the output in IXF format:

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

The following example shows how to export LOBs to an DEL file:

```
db2 export to myfile.del of del lobs to mylobs
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export LOBs to a DEL file, specifying a second directory for files that may not fit into the first directory:

```
db2 export to myfile.del of del
   lobs to /db2exp1, /db2exp2 modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export data to a DEL file, using a single quotation mark as the string delimiter, a semicolon as the column delimiter, and a comma as the decimal point. The same convention should be used when importing data back into the database:

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

API Examples

The following sample program shows how to:

- Export information from the STAFF table in the SAMPLE database to the file EXPTABLE.DEL.
- Import that information from the delimited text file to a new table, IMPTABLE.

For detailed information about the SAMPLE database, see the *Administration Guide*.

The source file for this sample program (impexp.sqc) can be found in the \sqllib\samples\c directory. It contains both DB2 APIs and embedded SQL calls. The script file bldvaemb.cmd, located in the same directory, contains the commands to build this and other sample programs. For general information about creating applications containing DB2 administrative APIs, and detailed information about compile and link options, see the *Application Building Guide*. To build the sample program impexp from the source file impexp.sqc on OS/2:

1. Copy the files impexp.sqc, bldvaemb.cmd, util.c, and util.h to a working directory.
2. If the database manager is not running, issue the command db2start.
3. Enter bldvaemb impexp sample. The following files are generated:

   ```
   impexp.bnd
   impexp.c
   util.obj
   impexp.obj
   impexp.exe
   ```

To run the sample program (executable file), enter impexp. You might find it useful to examine some of the generated files, such as the message file, and the delimited ASCII data file.

```
/*******************************************************************************
**
** Source File Name = impexp.sqc  1.4
**
**    PURPOSE :
**       This program is an example of how APIs are implemented in order to
**       export and import tables and table data.  The order of the program
**       is as follows:
**          - export a table to a comma-delimited text file
**          - import the comma-delimited text file to a DB2 table
**       This program needs the embedded SQL calls in order to connect to
**       an existing database, then to create a temporary table to work with.
**
**    STRUCTURES USED :
**       sqldcol
**       sqlchar
**       sqluexpt_out
**       sqluimp_in
**       sqluimp_out
**       sqlca
**
**    APIs USED :
**            IMPORT TO             sqluimpt_api
**            EXPORT                sqlgexpt
**
**    FUNCTIONS DECLARED :
**       'C' COMPILER LIBRARY :
**          stdio.h  -  printf
**          string.h -  fgets, strncpy
**
**       DBMS LIBRARY :
**          sqlenv.h -  see "APIs USED" above
**
**       OTHER :
**          internal :
**             list_dcs :         Displays a directory of databases
**
**          external :
**             check_error :     Checks for SQLCODE error, and prints out any
**             [in UTIL.C]       related information available.
**
**    EXTERNAL DEPENDANCIES :
**       - Ensure existence of database (SAMPLE) for precompile purposes.
**       - Precompile with the SQL precompiler (PREP in DB2)
**       - Bind to a database (BIND in DB2)
**       - Compile and link with the IBM Cset++ compiler (AIX and OS/2)
**         or the Microsoft Visual C++ compiler (Windows)
**         or the compiler supported on your platform.
**
```

```
*******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#ifndef DB2MAC
#include <malloc.h>
#endif
#include "util.h"

#ifdef DB268K
/* Need to include ASLM for 68K applications */
#include <LibraryManager.h>
#endif

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;

EXEC SQL INCLUDE SQLCA;
int main (int argc, char *argv[]) {
   short int           callerAction = 0;
   struct sqldcol      columnData;
   struct sqlchar      *columnStringPointer;
   struct sqluexpt_out  outputInfo;
   struct sqluimpt_in   impInput;
   struct sqluimpt_out  impOutput;

   char        datafile[] = "EXPTABLE.DEL";
   char        statement[] = "select name, id from staff";
   char        impStatement[] = "insert into imptable (name, id)";
   char        msgfile_x[] = "EXPMSG.TXT";
   char        msgfile_m[] = "IMPMSG.TXT";
   char        fileFormat[] = "DEL";

   EXEC SQL BEGIN DECLARE SECTION;
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

#ifdef DB268K
   /* Before making any API calls for 68K environment,
      need to initial the Library Manager */
InitLibraryManager(0,kCurrentZone,kNormalMemory);
atexit(CleanupLibraryManager);
#endif

   /* need to preset the size of structure field and counts */
   outputInfo.sizeOfStruct = SQLUEXPT_OUT_SIZE;
   impInput.sizeOfStruct = SQLUIMPT_IN_SIZE;
   impOutput.sizeOfStruct = SQLUIMPT_OUT_SIZE;
   impInput.restartcnt = impInput.commitcnt = 0;

   /*****************************************************************\
   * need to allocate the proper amount of space for the SQL statement *
   \*****************************************************************/
```

```
 columnStringPointer = (struct sqlchar *)malloc(strlen(statement)
    + sizeof (struct sqlchar));
 columnStringPointer->length = strlen(statement);
 strncpy (columnStringPointer->data, statement, strlen(statement));
 /* DELimited format can not have specified names, therefore the
    column method is 'D'efault */
 columnData.dcolmeth = 'D';

 if (argc == 1) {
    EXEC SQL CONNECT TO sample;
CHECKERR ("CONNECT TO SAMPLE");
 }
 else if (argc == 3) {
    strcpy (userid, argv[1]);
    strcpy (passwd, argv[2]);
    EXEC SQL CONNECT TO sample USER :userid USING :passwd;
    CHECKERR ("CONNECT TO SAMPLE");
 }
 else {
    printf ("\nUSAGE: impexp [userid passwd]\n\n");
    return 1;
 } /* endif */

 printf ("exporting NAME and ID from STAFF table into file '%s'\n", datafile);
 /*******************\
 * EXPORT API called *
 \*******************/
 sqluexpr (datafile, NULL, NULL, &columnData, columnStringPointer,
    fileFormat, NULL, msgfile_x, 0, &outputInfo, NULL, &sqlca);
 CHECKERR ("exporting table");
 printf ("rows exported %d\n", outputInfo.rowsExported);
 free (columnStringPointer);

 /******************************************************************\
 * need to allocate the proper amount of space for the SQL statement *
 \******************************************************************/
 columnStringPointer = (struct sqlchar *)malloc(strlen(impStatement)
    + sizeof (struct sqlchar));
 columnStringPointer->length = strlen(impStatement);
 strncpy (columnStringPointer->data, impStatement, strlen(impStatement));

 printf ("creating a temporary table 'imptable' to import into\n");
 /* create a temporary table to import into */
 EXEC SQL CREATE TABLE imptable (name VARCHAR(15), id INT);
 CHECKERR ("CREATE TABLE");

 printf ("importing the file '%s' into the 'imptable'\n", datafile);
 /*******************\
 * IMPORT API called *
 \*******************/
 sqluimpr (datafile, NULL, &columnData, columnStringPointer, fileFormat,
    NULL, msgfile_m, 0, &impInput, &impOutput, NULL, NULL, &sqlca);
 CHECKERR ("importing table");
 printf ("rows imported %d\nnumber of rows committed %d\n",
    impOutput.rowsInserted, impOutput.rowsCommitted);
```

```
   free (columnStringPointer);

   /* drop the table  */
   EXEC SQL DROP TABLE imptable;
   CHECKERR ("DROP TABLE");

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
}
/* end of program : impexp.sqc */
```

Restrictions

The following restriction applies to the export utility:

- This utility does not support the use of nicknames.

Troubleshooting

During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter.

These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Chapter 2. Import

This chapter describes the DB2 UDB import utility, which uses the SQL INSERT statement to write data from an input file into a table or view. If the target table or view already contains data, you can either replace or append to the existing data.

The following topics are covered:

For information about importing DB2 Data Links Manager data, see “Using Import to Move DB2 Data Links Manager Data” on page 152. For information about importing data from typed tables, see “Moving Data Between Typed Tables” on page 165. For information about importing data from a file on the DB2 Connect workstation to a DRDA server database, and the reverse, see “Moving Data With DB2 Connect” on page 163.

Import Overview

The import utility inserts data from an input file into a table or updatable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

The following information is required when importing data:

- The path and the name of the input file.
- The name or alias of the target table or view.
- The format of the data in the input file. This format can be IXF, WSF, DEL, or ASC. See “Appendix C. Export/Import/Load Utility File Formats” on page 179.
- Whether the input data is to be inserted into the table or view, or whether existing data in the table or view is to be updated or replaced by the input data.
- A message file name, if the utility is invoked through the application programming interface (API), **sqluimpr**.
- When working with typed tables, you may need to provide the method or order by which to progress through all of the structured types. The order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy is called the *traverse* order. This order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

  When working with typed tables, you may also need to provide the subtable list. This list shows into which subtables and attributes to import data.

  For more information, see “Moving Data Between Typed Tables” on page 165.

You can also specify:

- The method to use for importing the data: column location, column name, or relative column position.
- The number of rows to INSERT before committing the changes to the table. Requesting periodic COMMITs reduces the number of rows that are lost if a failure and a ROLLBACK occur during the import operation. It also prevents the DB2 logs from getting full when processing a large input file.
- The number of file records to skip before beginning the import operation. If an error occurs, you can restart the import operation immediately following the last row that was successfully imported and committed.
- The names of the columns within the table or view into which the data is to be inserted.

Privileges, Authorities, and Authorization Required to Use Import

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

To use the import utility to create a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database. To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view. To append data to an existing table or view, you must have SELECT and INSERT privileges for the table or view.

Using Import

Before Using Import

Before invoking the import utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be imported. Since the utility will issue a COMMIT or a ROLLBACK statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking import.

Invoking Import

The import utility can be invoked through:

- The command line processor (CLP).

  Following is an example of the IMPORT command issued through the CLP:

  ```
  db2 import from stafftab.ixf of ixf insert into userid.staff
  ```

- The Import notebook in the Control Center. To open the Import notebook:

  1. From the Control Center, expand the object tree until you find the Tables folder.
  2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
  3. Click the right mouse button on the table you want in the contents pane, and select Import from the pop-up menu. The Import notebook opens.

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.

- An application programming interface (API), **sqluimpr**. For information about this API, see “Import API” on page 36. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

Using Import in a Client/Server Environment

When you import a file to a remote database, a stored procedure can be called to perform the import on the server. A stored procedure will not be called when:

- The application and database code pages are different.
- The file being imported is a multiple-part PC/IXF file.
- The method used for importing the data is either column name or relative column position.
- The target column list provided is longer than 4KB.
- An OS/2 or DOS client is importing a file from diskette.
- The LOBS FROM clause or the `lobsinfile` modifier is specified.
- The NULL INDICATORS clause is specified for ASC files.

When import uses a stored procedure, messages are created in the message file using the default language installed on the server. The messages are in the language of the application if the language at the client and the server are the same.

The import utility creates two temporary files in the `tmp` subdirectory of the `sqllib` directory (or the directory indicated by the **DB2INSTPROF** registry variable, if specified). One file is for data, and the other file is for messages generated by the import utility.

If you receive an error about writing or opening data on the server, ensure that:

- The directory exists.
- There is sufficient disk space for the files.
- The instance owner has write permission in the directory.

Using Import with Buffered Inserts

In a partitioned database environment, the import utility can be enabled to use buffered inserts. This reduces the messaging that occurs when data is imported, resulting in better performance; however, since details about a failed buffered insert are not returned, this option should only be enabled if you are not concerned about error reporting.

Use the DB2 bind utility to request buffered inserts. The import package, db2uimpm.bnd, must be rebound against the database using the INSERT BUF option. For example:

```
db2 connect to your_database
db2 bind db2uimpm.bnd insert buf
```

Note: The buffered inserts feature is disabled during any import operation in which the INSERT_UPDATE parameter is specified.

Recreating an Exported Table

You can use the import utility to recreate a table that was saved through the export utility. The table must have been exported to an IXF file, and the SELECT statement used during the export operation must have met certain conditions (for example, no column names can be used in the SELECT clause; only `select *` is permitted). When creating a table from an IXF file, not all attributes of the original table are preserved. For example, referential constraints, foreign key definitions, and user-defined data types are not retained. The following attributes of the original table are retained:

- Column information:
  - Names.
  - Types, including user-defined distinct types, which are preserved as their base type.
  - Lengths (except for lob_file types).
  - Code pages (if applicable).
- Index information:
  - Name.
  - Creator.
  - Column names of key parts (with a restriction if + or − is in the names).
  - Ascending or descending.
  - Uniqueness.

Importing Large Objects (LOBs)

When importing into large object (LOB) columns, the data can come either from the same file as the rest of the column data, or from separate files. In the latter case, there is one file for each LOB instance.

The column in the main input data file contains either the import data (default), or the name of a file where the import data is stored.

Importing Large Objects (LOBs)

Notes:

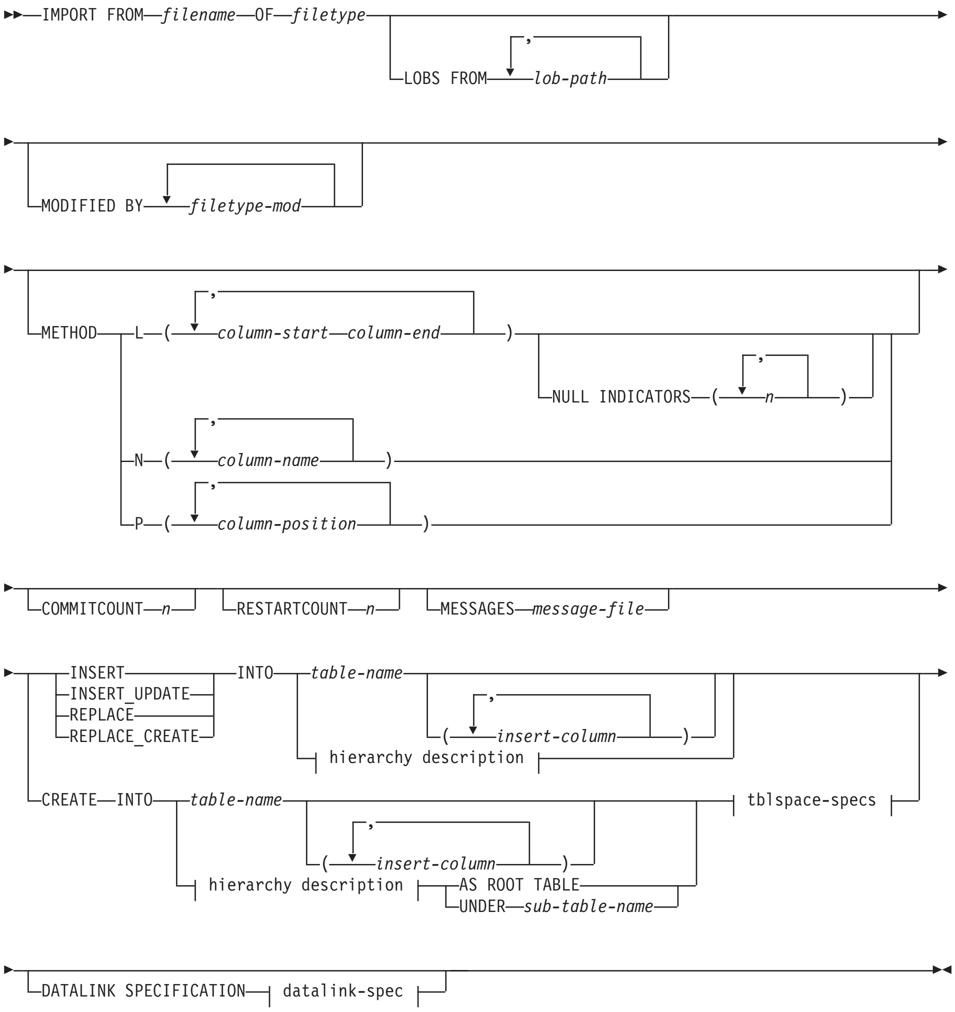
1. When LOB data is stored in the main input data file, no more than 32KB of data is allowed. Truncation warnings are ignored.
2. All of the LOB data must be stored in the main file, or each LOB is stored in separate files. The main file cannot have a mixture of LOB data and file names. LOB values are imported from separate files by using the `lobsinfile` modifier (see “File Type Modifiers (Import)” on page 49), and the `LOBS FROM` clause (see “IMPORT Command” on page 29).

Importing User-defined Distinct Types (UDTs)

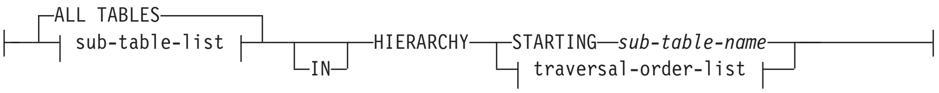
The import utility casts user-defined distinct types (UDTs) to similar base data types automatically. This saves you from having to explicitly cast UDTs to the base data types. Casting allows for comparisons between UDTs and the base data types in SQL.

IMPORT Command

Command Syntax



hierarchy description:



sub-table-list:

```
     .-,--------------------------------------------------.
     V                                                    |
|--(----sub-table-name--+-------------------------------+--+--)--------------------|
                        |        .-,------------.       |
                        |        V              |       |
                        '-(--------insert-column-+--)----'
```

traversal-order-list:

```
     .-,-------------------.
     V                     |
|--(----sub-table-name-----+--)----------------------------------------------------|
```

tblspace-specs:

```
|--+----------------------------------------------------------------------------------+--|
   '-IN--tablespace-name--+-------------------------------+--+----------------------------+-'
                          '-INDEX IN--tablespace-name-----'  '-LONG IN--tablespace-name---'
```

datalink-spec:

```
   .-,-------------------------------------------------------------------------------------.
   V                                                                                       |
|----(--+--------------------+--+------------------------------------+--+--------------------------+--)--+--|
        '-DL_LINKTYPE URL----'  +-DL_URL_REPLACE_PREFIX--"prefix"----+  '-DL_URL_SUFFIX--"suffix"--'
                                '-DL_URL_DEFAULT_PREFIX--"prefix"----'
```

Command Parameters

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT n
Performs a COMMIT after every *n* records are imported.

CREATE
Creates the table definition and row contents. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

Note: If the data was exported from an MVS host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are less than 254, CREATE may fail because the rows are too long. In this case, the table should be created manually, and IMPORT with INSERT should be invoked, or, alternatively, the LOAD command should be used.

DATALINK SPECIFICATION

For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DL_LINKTYPE

If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″prefix″

If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification.

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
```

If no prefix is found in a column's data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix "http://toronto":

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.
- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX ″prefix″

This clause is useful for loading or importing data previously generated by the export utility, when the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://toronto/a/b/c″. Note that ″toronto″ replaces ″coyote″.
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX ″suffix″

If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the ″path″ component of the URL part of the DATALINK value.

FROM filename

Specifies the file that contains the data to be imported. If the path is omitted, the current working directory is used.

HIERARCHY

Specifies that hierarchical data is to be imported.

IN tablespace-name

Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN tablespace-name

Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The specified table space must exist, and must be a REGULAR DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column

Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO table-name
Specifies the database table into which the data is to be imported. This table cannot be a system table or a summary table.

One can use an alias for INSERT, INSERT_UPDATE, or REPLACE, except in the case of a down-level server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM lob-path
Specifies one or more paths that store LOB files. The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. This option is ignored if the `lobsinfile` modifier is not specified.

LONG IN tablespace-name
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the IN clause is a DMS table space. The table space must exist, and must be a LONG DMS table space.

MESSAGES message-file
Specifies the destination for warning and error messages that occur during an import operation. If the file already exists, the import utility appends the information. If the complete path to the file is not specified, the utility uses the current directory and the default drive as the destination. If *message-file* is omitted, the messages are written to standard output.

METHOD

L Specifies the start and end column numbers from which to import data.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns to be imported.

Note: This method can only be used with IXF files.

P Specifies the numbers of the columns to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY filetype-mod
Specifies additional options (see Table 5 on page 49).

NULL INDICATORS n
Specifies (by number) one or more columns in the data file that are to be used as null indicator fields. If this option is used, a null indicator column for each data column must be specified. Zero (0) indicates that the data column is not nullable, and that there will always be data in that column.

While processing each row, a Y indicates that the column data is NULL, while an N indicates that the column data is not NULL, and that column data specified by the METHOD L option will be imported.

OF filetype
Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony
- IXF (integrated exchange format, PC version), which means it was exported from the same or another DB2 table. An IXF file also contains the table definition and definitions of any existing indexes, except when columns are specified in the SELECT statement.

For more information about file formats, see “Appendix C. Export/Import/Load Utility File Formats” on page 179.

REPLACE
Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. It is not valid for tables with DATALINK columns. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

REPLACE_CREATE
If the table exists, deletes all existing data from the table by truncating

the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents.

This option can only be used with IXF files. It is not valid for tables with DATALINK columns. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

RESTARTCOUNT n

Specifies that an import operation is to be started at record *n* + 1. The first *n* records are skipped.

STARTING sub-table-name

A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list

For typed tables with the INSERT or the INSERT_UPDATE option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list

For typed tables with the INSERT, INSERT_UPDATE, or the REPLACE option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER sub-table-name

Specifies a parent table for creating one or more sub-tables.

Import API

C API Syntax

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqluimpr (
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Import */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgimpr (
    unsigned short DataFileNameLen,
    unsigned short FileTypeLen,
    unsigned short MsgFileNameLen,
    char * pDataFileName,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pMsgFileName,
    short CallerAction,
    struct sqluimpt_in*  pImportInfoIn,
    struct sqluimpt_out*  pImportInfoOut,
    long * NullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

DataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the input file name.

FileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the input file type.

MsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

pDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

pLobPathList
Input. An *sqlu_media_list* using *media_type* `SQLU_LOCAL_MEDIA`, and the *sqlu_media_entry* structure listing paths on the client where the LOB files can be found.

For more information, see "SQLU-MEDIA-LIST " in the *Administrative API Reference*.

pDataDescriptor
Input. Pointer to an *sqldcol* structure containing information about the columns being selected for import from the external file. The value of the *dcolmeth* field determines how the remainder of the information provided in this parameter is interpreted by the import utility. Valid values for this parameter (defined in `sqlutil`) are:

SQL_METH_N
Names. Selection of columns from the external input file is by column name.

SQL_METH_P
Positions. Selection of columns from the external input file is by column position.

SQL_METH_L
Locations. Selection of columns from the external input file is by column location. The database manager rejects an import call with a location pair that is invalid because of any one of the following conditions:

- Either the beginning or the ending location is not in the range from 1 to the largest signed 2-byte integer.
- The ending location is smaller than the beginning location.

- The input column width defined by the location pair is not compatible with the type and the length of the target column.

A location pair with both locations equal to zero indicates that a nullable column is to be filled with NULLs.

SQL_METH_D

Default. If *pDataDescriptor* is NULL, or is set to SQL_METH_D, default selection of columns from the external input file is done. In this case, the number of columns and the column specification array are both ignored. The first *n* columns of data in the external input file are taken in their natural order, where *n* is the number of database columns into which the data is to be imported.

For more information, see “SQLDCOL” in the *Administrative API Reference*.

pActionString

Input. Pointer to an *sqlchar* structure containing a 2-byte long field, followed by an array of characters identifying the columns into which data is to be imported.

The character array is of the form:

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
[DATALINK SPECIFICATION datalink-spec]
```

INSERT

Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE

Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

REPLACE

Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and replaced if indexixf is in *FileTypeMod*, and *FileType* is

SQL_IXF.) If the table is not already defined, an error is returned. **Attention:** If an error occurs after the existing data is deleted, that data is lost.

CREATE
Creates the table definition and the row contents using the information in the specified PC/IXF file, if the specified table is not defined. If the file was previously exported by DB2, indexes are also created. If the specified table is already defined, an error is returned. This option is valid for the PC/IXF file format only.

REPLACE_CREATE
Replaces the table contents using the PC/IXF row information in the PC/IXF file, if the specified table is defined. If the table is not already defined, the table definition and row contents are created using the information in the specified PC/IXF file. If the PC/IXF file was previously exported by DB2, indexes are also created. This option is valid for the PC/IXF file format only. **Attention:** If an error occurs after the existing data is deleted, that data is lost.

tname The name of the table, typed table, view, or object view into which the data is to be inserted. An alias for REPLACE, INSERT_UPDATE, or INSERT can be specified, except in the case of a down-level server, when a qualified or unqualified name should be specified. If it is a view, it cannot be a read-only view.

tcolumn-list
A list of table or view column names into which the data is to be inserted. The column names must be separated by commas. If column names are not specified, column names as defined in the CREATE TABLE or the ALTER TABLE statement are used. If no column list is specified for typed tables, data is inserted into all columns within each sub-table.

sub-table-name
Specifies a parent table when creating one or more sub-tables under the CREATE option.

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the *traversal-order-list*.

HIERARCHY
Specifies that hierarchical data is to be imported.

STARTING
Keyword for hierarchy only. Specifies that the default order, starting from a given sub-table name, is to be used.

UNDER
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created under a given sub-table.

AS ROOT TABLE
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created as a stand-alone hierarchy.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the IMPORT command (see “IMPORT Command” on page 29).

The *tname* and the *tcolumn-list* parameters correspond to the *tablename* and the *colname* lists of SQL INSERT statements, and have the same restrictions.

The columns in *tcolumn-list* and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the *sqldcol* structure is inserted into the table or view field corresponding to the first element of the *tcolumn-list*).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

For more information, see “SQLCHAR” in the *Administrative API Reference*.

pFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred method for exporting data from a table so that it can be imported later into the same table or into another database manager table.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

For more information about file formats, see "Appendix C. Export/Import/Load Utility File Formats" on page 179.

pFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see "SQLCHAR" in the *Administrative API Reference*, and "File Type Modifiers (Import)" on page 49.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, it is appended to. If it does not exist, a file is created.

CallerAction
Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested import operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action

requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

pImportInfoIn
Input. Optional pointer to the *sqluimpt_in* structure containing additional input parameters. For information about this structure, see "SQLUIMPT-IN Data Structure" on page 46.

pImportInfoOut
Output. Optional pointer to the *sqluimpt_out* structure containing additional output parameters. For information about this structure, see "SQLUIMPT-OUT Data Structure" on page 47.

NullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. Therefore, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is NULL, and `N` indicates that the table column data is not NULL.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see "SQLCA" in the *Administrative API Reference*.

REXX API Syntax

```
IMPORT FROM datafile OF filetype
[MODIFIED BY :filetmod]
[METHOD {L|N|P} USING :dcoldata]
[COMMITCOUNT :commitcnt] [RESTARTCOUNT :restartcnt]
MESSAGES msgfile
{INSERT|REPLACE|CREATE|INSERT_UPDATE|REPLACE_CREATE}
INTO tname [(:columns)]
[OUTPUT INTO :output]

CONTINUE IMPORT

STOP IMPORT
```

REXX API Parameters

datafile
Name of the file from which the data is to be imported.

filetype
The format of the data in the external import file. The supported file formats are:

DEL Delimited ASCII

ASC Non-delimited ASCII

WSF Worksheet format

IXF PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options (see “File Type Modifiers (Import)” on page 49).

L|N|P
A character specifying the method to be used to select columns within the external input file. Valid values are:

L Location

N Name

P Position.

dcoldata
A compound REXX host variable containing information about the columns selected for import from the external input file. The content of the structure depends upon the specified *method*. In the following, XXX represents the name of the host variable:

- Location method

  XXX.0 Number of elements in the remainder of the variable

XXX.1 A number representing the starting location of this column in the input file. This column becomes the first column in the database table.

XXX.2 A number representing the ending location of the column.

XXX.3 A number representing the starting location of this column in the input file. This column becomes the second column in the database table.

XXX.4 A number representing the ending location of the column.

XXX.5 and so on.

- Name method

  XXX.0 Number of column names contained in the host variable.

  XXX.1 First name.

  XXX.2 Second name.

  XXX.3 and so on.

- Position method

  XXX.0 Number of column positions contained in the host variable.

  XXX.1 A column position in the external input file.

  XXX.2 A column position in the external input file.

  XXX.3 and so on.

tname Name of the target table or view. Data cannot be imported to a read-only view.

columns
A REXX host variable containing the names of the columns in the table or the view into which the data is to be inserted. In the following, XXX represents the name of the host variable:

XXX.0 Number of columns.

XXX.1 First column name.

XXX.2 Second column name.

XXX.3 and so on.

msgfile
File, path, or device name where error and warning messages are to be sent.

commitcnt
Performs a COMMIT after every *commitcnt* records are imported.

restartcnt

Specifies that an import operation is to be started at record *restartcnt* + 1. The first *restartcnt* records are skipped.

output

A compound REXX host variable into which information from the import operation is passed. In the following, XXX represents the name of the host variable:

XXX.1 Number of records read from the external input file during the import operation.

XXX.2 Number of records skipped before inserting or updating begins.

XXX.3 Number of rows inserted into the target table.

XXX.4 Number of rows in the target table updated with information from the imported records.

XXX.5 Number of records that could not be imported.

XXX.6 Number of records imported successfully and committed to the database, including rows inserted, updated, skipped, and rejected.

SQLUIMPT-IN Data Structure

This structure is used to pass information to the “Import API” on page 36.

Table 3. Fields in the SQLUIMPT-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
COMMITCNT	INTEGER	The number of records to import before committing them to the database. A COMMIT is performed whenever *commitcnt* records are imported.
RESTARTCNT	INTEGER	The number of records to skip before starting to insert or update records. This parameter should be used if a previous attempt to import records fails after some records have been committed to the database. The specified value represents a starting point for the next import operation.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-IN */
/* ... */
SQL_STRUCTURE sqluimpt_in
{
  unsigned long   sizeOfStruct;
  unsigned long   commitcnt;
  unsigned long   restartcnt;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-IN.
    05 SQL-SIZE-OF-UIMPT-IN   PIC 9(9) COMP-5 VALUE 12.
    05 SQL-COMMITCNT          PIC 9(9) COMP-5 VALUE 0.
    05 SQL-RESTARTCNT         PIC 9(9) COMP-5 VALUE 0.
*
```

SQLUIMPT-OUT Data Structure

This structure is used to pass information from the “Import API” on page 36.

Table 4. Fields in the SQLUIMPT-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	INTEGER	Size of this structure in bytes.
ROWSREAD	INTEGER	Number of records read from the file during import.
ROWSSKIPPED	INTEGER	Number of records skipped before inserting or updating begins.
ROWSINSERTED	INTEGER	Number of rows inserted into the target table.
ROWSUPDATED	INTEGER	Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).
ROWSREJECTED	INTEGER	Number of records that could not be imported.
ROWSCOMMITTED	INTEGER	Number of records imported successfully and committed to the database.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLUIMPT-OUT */
/* ... */
SQL_STRUCTURE sqluimpt_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsRead;
  unsigned long   rowsSkipped;
  unsigned long   rowsInserted;
  unsigned long   rowsUpdated;
  unsigned long   rowsRejected;
  unsigned long   rowsCommitted;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQL-UIMPT-OUT.
    05 SQL-SIZE-OF-UIMPT-OUT  PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWSREAD           PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSSKIPPED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSINSERTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSUPDATED        PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSREJECTED       PIC 9(9) COMP-5 VALUE 0.
    05 SQL-ROWSCOMMITTED      PIC 9(9) COMP-5 VALUE 0.
*
```

File Type Modifiers (Import)

Table 5. Valid File Type Modifiers (Import)

Modifier	Description
All File Formats	
compound=*x*	*x* is a number between 1 and 100 inclusive (7 on DOS/Windows). Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time.
lobsinfile	*lob-path* specifies the path to the files containing LOB values.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If the column is defaultable, the default value is loaded • If the column is nullable and not defaultable, a NULL is loaded • If the column is not nullable and not defaultable, an error is returned, and the utility stops processing.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: ",," is specified for the column • For ASC files: The NULL indicator is set to yes for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.
ASC (Non-delimited ASCII) File Format	
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[b] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the null indicator character is specified to be the letter `N`, then `n` is also recognized as a null indicator.
reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.

striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept.
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`modified by striptblanks`
	`method l (1 10, 12 15) messages msgs.txt`
	`insert into staff`
	This option cannot be specified together with `striptnulls`. These are mutually exclusive options.
	Note: This option replaces the obsolete `t` option, which is supported for back-level compatibility only.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	Description
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.
DEL (Delimited ASCII) File Format	

chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[ab]
	The single quotation mark (') can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`   modified by chardel''`
	`   method p (1, 4) insert into staff (id, years)"`

coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[ab]
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`   modified by coldel;`
	`   messages msgs.txt insert into staff`

datesiso	Date format. Causes all date data values to be imported in ISO format.
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**

decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[ab]
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`   modified by chardel''`
	`   decpt; messages msgs.txt insert into staff"`

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 import ... modified by delprioritychar ...`
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [ab] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
nodoubledel	Suppresses recognition of double character delimiters.
IXF File Format	

Table 5. Valid File Type Modifiers (Import) (continued)

Modifier	**Description**
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully imported if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Notes:

1. The import utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the import operation fails, and an error code is returned.
2. [a] “Delimiter Restrictions” on page 16 lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] The character must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```

Character Set and NLS Considerations

Unequal code page situations, involving expansion or contraction of the character data, can sometimes occur. For example, Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) may have different length encodings for the same character. Normally, comparison of input data length to target column length is performed before reading in any data. If the input length is greater than the target length, NULLs are inserted into that column if it is nullable. Otherwise, the request is rejected. If the nochecklengths modifier (see “File Type Modifiers (Import)” on page 49) is specified, no initial comparison is performed, and an attempt is made to import the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is imported.

Example Import Sessions

CLP Examples

The following example shows how to import information from myfile.ixf to the STAFF table:

```
   db2 import from myfile.ixf of ixf messages msg.txt insert into staff
```

```
SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".

SQL3153N  The T record in the PC/IXF file has name "myfile", qualifier "        ",
and source "            ".

SQL3109N  The utility is beginning to load data from file "myfile".

SQL3110N  The utility has completed processing.  "58" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

The following example shows how to import the table MOVIETABLE from the input file delfile1, which has data in the DEL format:

```
db2 import from delfile1 of del
    modified by dldel|
    insert into movietable (actorname, description, url_making_of, url_movie)
    datalink specification (dl_url_default_prefix "http://narang"),
    (dl_url_replace_prefix "http://bomdel" dl_url_suffix ".mpeg")
```

Notes:

1. The table has four columns:

   ```
   actorname            VARCHAR(n)
   description          VARCHAR(m)
   url_making_of        DATALINK (with LINKTYPE URL)
   url_movie            DATALINK (with LINKTYPE URL)
   ```

2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.

API Examples

See “API Examples” on page 18.

Optimizing Import Performance

Specifying target table column names or a specific importing method makes importing to a remote database slower.

Restrictions and Limitations

The following restrictions apply to the import utility:

- This utility does not support the use of nicknames.
- If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only appended to.
- You cannot perform an import replace operation into an underlying table of a summary table defined in refresh immediate mode.
- You cannot import data into a system table or a summary table.
- Views cannot be created through the import utility.

Import Restrictions and Limitations

- Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)
- Because the import utility generates its own SQL statements, the maximum statement size of 64KB may, in some cases, be exceeded.

The following limitations apply to the import utility:

- If an import operation is run against a remote database, and the output message file is very long (more than 60KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB of message information are always retained.

Troubleshooting

During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter.

These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Chapter 3. Load

This chapter describes the DB2 UDB load utility, which moves data from files, named pipes, or devices into a DB2 table. These data sources must reside on the node where the database resides, and the table must exist. If the table receiving the new data already contains data, you can replace or append to the existing data.

The following topics are covered:

- “Restrictions and Limitations” on page 130
- “Troubleshooting” on page 130.

For information about loading DB2 Data Links Manager data, see “Using Load to Move DB2 Data Links Manager Data” on page 153.

Load Overview

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data. The utility can handle all data types, including large objects (LOBs) and user-defined types (UDTs). The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes). The data being loaded must be local to the server (unlike import and export, which support the passing of data from the client). For a detailed comparison of the load and the import utilities, see “Appendix B. Differences Between the Import and the Load Utility” on page 177.

The load process consists of three distinct phases (see Figure 1):

- Load, during which the data is written to the table.

  During the load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. Save points, or points of consistency, are established at intervals specified through the SAVECOUNT parameter in the LOAD command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point. For DATALINK columns defined with FILE LINK CONTROL, link operations are performed for non-NULL column values. If a failure occurs, you can restart the load operation; the RESTART option automatically restarts the load operation from the last successful consistency point. The TERMINATE option rolls back the failed load operation.

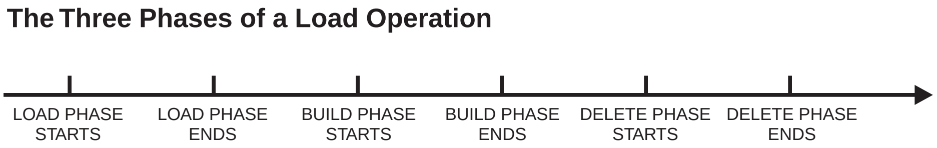


Figure 1. The Three Phases of the Load Process: Load, Build, and Delete.. Associated table spaces are in load pending state from the beginning of the load phase until the end of the build phase, and in delete pending state from the end of the build phase until the end of the delete phase.

- Build, during which indexes are created.

During the build phase, indexes are created based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (if the STATISTICS YES with INDEXES option was specified). The statistics are similar to those collected through the RUNSTATS command (see the *Command Reference*). If a failure occurs during the build phase, the RESTART option automatically restarts the load operation at the appropriate point.

Unique key violations are placed into the exception table, if one was specified (see “Exception Table” on page 112), and messages about rejected rows are written to the message file. Following the completion of the load process, review these messages, resolve any problems, and insert corrected rows into the table.

- Delete, during which the rows that caused a unique key violation or a DATALINK violation are removed from the table.

  Do not attempt to delete or to modify any temporary files created by the load utility. Some temporary files are critical to the delete phase. If a failure occurs during the delete phase, the RESTART option automatically restarts the load operation at the appropriate point.

  Note: Each deletion event is logged. If you have a large number of records that violate the uniqueness condition, the log could fill up during the delete phase.

The following information is required when loading data:

- The path and the name of the input file, named pipe, or device.
- The name or alias of the target table.
- The format of the data in the input file. This format can be DEL, ASC, or PC/IXF. See “Appendix C. Export/Import/Load Utility File Formats” on page 179.
- Whether the input data is to be appended to the table, or is to replace the existing data in the table.
- A message file name, if the utility is invoked through the application programming interface (API), **sqluload**.

You can also specify:

- The method to use for loading the data: column location, column name, or relative column position.
- How often the utility is to establish consistency points. Use the SAVECOUNT parameter to specify this value. If this parameter is specified, a load restart operation will start at the last consistency point, instead of at the beginning.
- The names of the table columns into which the data is to be inserted.

- The paths and the names of the input files in which LOBs are stored. The lobsinfile modifier tells the load utility that all LOB data is being loaded from files (see “File Type Modifiers (Load)” on page 104).
- Whether column values being loaded have implied decimal points. The implieddecimal modifier tells the load utility that decimal points are to be applied to the data as it enters the table (see “File Type Modifiers (Load)” on page 104). For example, the value 12345 is loaded into a DECIMAL(8,2) column as 123.45, not 12345.00.
- Whether the utility should modify the amount of free space available after a table is loaded. Additional free space permits INSERT and UPDATE growth to the table following the completion of a load operation. Reduced free space keeps related rows more closely together and can enhance table performance.
  - The totalfreespace modifier enables you to append empty data pages to the end of the loaded table. The number specified with this parameter is the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if you specified the number twenty with this parameter, and the table has 100 data pages, twenty additional empty pages are appended. The total number of data pages in the table will then be 120.
  - The pagefreespace modifier enables you to control the amount of free space that will be allowed on each loaded data page. The number specified with this parameter is the percentage of each data page that is to be left as free space. The first row in a page is added without restriction. Therefore, with very large rows and a large number specified with this parameter, there may be less free space left on each page than that indicated by the value specified with this parameter.
  - The indexfreespace modifier enables you to control the amount of free space that will be allowed on each loaded index page. The number specified with this parameter is the percentage of each index page that is to be left as free space. The first index entry in a page is added without restriction. Additional index entries are placed in the index page, provided the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time. The indexfreespace value takes precedence over the PCTFREE value specified in the CREATE INDEX statement.

If you specify the pagefreespace modifier, and you have an index on the table, you might consider specifying indexfreespace. When deciding on the amount of free space to leave for each, consider that the size of each row being inserted into the table will likely be larger than the size of the associated key to be inserted into the index. In addition, the page size of the table spaces for the table and the index may be different.

- Whether statistics are to be gathered during the load process. This option is only supported if the load operation is running in REPLACE mode.

  If data is appended to a table, statistics are not collected. To collect current statistics on an appended table, invoke the runstats utility following completion of the load process. If gathering statistics on a table with a unique index, and duplicate keys are deleted during the delete phase, statistics are not updated to account for the deleted records. If you expect to have a significant number of duplicate records, do not collect statistics during the load operation. Instead, invoke the runstats utility following completion of the load process.
- Whether to keep a copy of the changes made. This is done to enable rollforward recovery of the database. This option is not supported if forward log recovery is disabled for the database; that is, if the database configuration parameters *logretain* and *userexit* are disabled. If no copy is made, and forward log recovery is enabled, the table space is left in backup pending state at the completion of the load operation (see “Pending States After a Load Operation” on page 124).

  Logging is required for fully recoverable databases. The load utility almost completely eliminates the logging associated with the loading of data. In place of logging, you have the option of making a copy of the loaded portion of the table. For information about how DB2 keeps tracks of the load copies, see “Using the Load Copy Location File” on page 67. If you have a database environment that allows for database recovery following a failure, you can do one of the following:

  – Explicitly request that a copy of the loaded portion of the table be made.

  – Take a backup of the table spaces in which the table resides immediately after the completion of the load operation.

If you are loading a table that already contains data, and the database is non-recoverable, ensure that you have a backed-up copy of the database, or the table spaces for the table being loaded, before invoking the load utility, so that you can recover from errors.

If you want to perform a sequence of multiple load operations on a recoverable database, the sequence of operations will be faster if you specify each load operation to be non-recoverable, and take a backup at the end of the load sequence, than if you invoke each of the load operations with the COPY YES option. You can use the NONRECOVERABLE option to specify that a load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped (see Figure 2 on page 62). With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation.

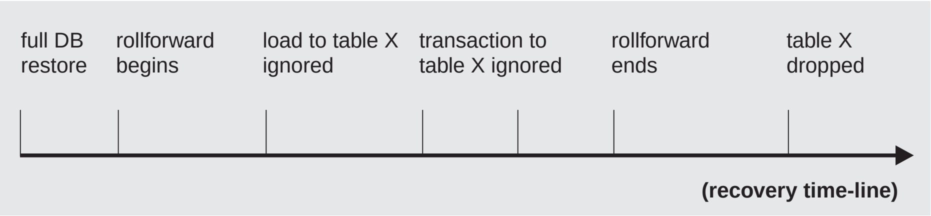


Figure 2. Non-recoverable Processing During a Roll Forward Action

For more information, see the database recovery chapter in the *Administration Guide*.

- The fully qualified path to be used when creating temporary files during a load operation. The name is specified by the TEMPFILES PATH parameter of the LOAD command. The default value is the database path. The path resides on the server machine, and is accessed by the DB2 instance exclusively. Therefore, any path name qualification given to this parameter must reflect the directory structure of the server, not the client, and the DB2 instance owner must have read and write permission on the path. This is true even if you are the instance owner. If you are not the instance owner, you must specify a location that is writable by the instance owner. For more information about temporary files, see “Load Temporary Files” on page 113.

Parallelism and Loading

The load utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment. There are several ways in which parallel processing of large amounts of data can take place using the load utility. One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation (see Figure 3 on page 63). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 4 on page 63). Both can be used together to provide even faster loading of data. For more information, see “Optimizing Load Performance” on page 125.

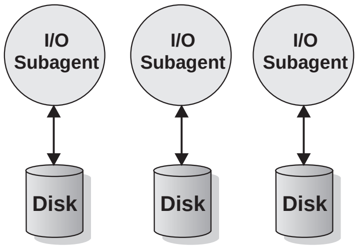


Figure 3. Taking Advantage of I/O Parallelism When Loading Data

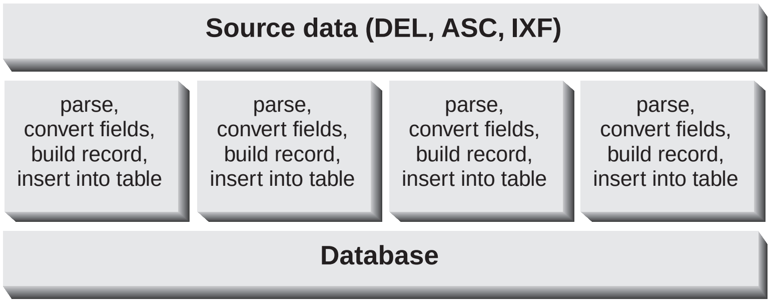

Source data (DEL, ASC, IXF)
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
Database

Figure 4. Taking Advantage of Intra-partition Parallelism When Loading Data

Privileges, Authorities, and Authorization Required to Use Load

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

To load data into a table, you must have one of the following:

- SYSADM authority
- DBADM authority
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in APPEND, TERMINATE, or RESTART mode
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode.

Authorities Required to Use Load

Since all load processes (and all DB2 server processes, in general), are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

If DB2 for Windows NT has been defined as a Service to the Windows NT operating system, the Service must have a user account with the required read / write file permissions to use LAN resources (drives, directories, and files).

Using Load

Before Using Load

Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking load.

Since data is loaded in the sequence that appears in the input file, if a particular sequence is desired, the data should be sorted before a load operation is attempted.

If clustering is required, the data should be sorted on the clustering index prior to loading.

Invoking Load

The load utility can be invoked through:

- The command line processor (CLP).

  Following is an example of the LOAD command issued through the CLP:

  ```
  db2 load from stafftab.ixf of ixf messages staff.msgs
     insert into userid.staff copy yes use adsm data buffer 4000
  ```

  In this example:

  – Any warning or error messages are placed in the `staff.msgs` file.

  – A copy of the changes made is stored in ADSTAR Distributed Storage Manager (ADSM). For more information about ADSM, see the “ADSTAR Distributed Storage Manager” section of the database recovery chapter in the *Administration Guide.*

  – Four thousand pages of buffer space are to be used during the load operation.

Following is another example of the LOAD command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   tempfiles path /u/myuser replace into staff
```

In this example:

– The table data is being replaced.

– The TEMPFILES PATH parameter is used to specify /u/myuser as the server path into which temporary files will be written.

Note: These examples use relative path names for the load input file. Relative path names are only allowed on calls from a client on the same node as the database. The use of fully qualified path names is recommended.

- The Load notebook in the Control Center. To open the Load notebook:
  1. From the Control Center, expand the object tree until you find the Tables folder.
  2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
  3. Click the right mouse button on the table you want in the contents pane, and select Load from the pop-up menu. The Load notebook opens.

  For general information about the Control Center, see the *Administration Guide*. Detailed information is provided through the online help facility within the Control Center.
- An application programming interface (API), **sqluload**. For information about this API, see “Load API” on page 84. For general information about creating applications containing DB2 administrative APIs, see the *Application Building Guide*.

Checking for Constraints Violations

Following a load operation, the loaded table may be in check pending state if it has table check constraints or referential integrity constraints defined on it. The STATUS flag of the SYSCAT.TABLES entry corresponding to the loaded table indicates the check pending state of the table. For the loaded table to be usable, the STATUS must have a value of N, indicating a normal state.

To remove the check pending state, use the SET INTEGRITY statement (see the *SQL Reference*). The SET INTEGRITY statement checks a table for constraints violations, and takes the table out of check pending state. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement will by default incrementally process the constraints (that is, it will check only the appended portion of the table for constraints violations). For example:

```
db2 load from infile1.ixf of ixf insert into table1
db2 set integrity for table1 immediate checked
```

Only the appended portion of TABLE1 is checked for constraint violations. Checking only the appended portion for constraints violations is faster than checking the entire table, especially in the case of a large table with small amounts of appended data.

One or more tables can be checked in a single invocation of this statement. If a dependent table is to be checked, the parent table must not be in check pending state. In the case of a referential integrity cycle, all the tables involved in the cycle must be included in a single invocation of the SET INTEGRITY statement. It may be convenient to check the parent table for constraints violations while a dependent table is being loaded. This can only occur if the two tables are not in the same table space.

When issuing the SET INTEGRITY statement, you can specify the INCREMENTAL option to explicitly request incremental processing. In most cases, this option is not needed, because the default behavior is incremental processing. If incremental processing is not possible, full processing is used automatically. When the INCREMENTAL option is specified, but incremental processing is not possible, an error is returned if:

- New constraints have been added to the table or to its parent table while both tables are in check pending state.
- A load replace operation has taken place.
- A parent table has been checked for integrity non-incrementally.

If the T table has one or more W values in the CONST_CHECKED column of the SYSCAT.TABLES catalog (see the *SQL Reference*, SET INTEGRITY statement, for a description of the W state), the system checks the entire table for constraints violations if the INCREMENTAL option is not specified. If the option *is* specified, it is allowed, but the CONST_CHECKED column of SYSTABLES will be marked as U to indicate that not all data has been verified by the system.

Use the load exception table option to capture information about rows with constraints violations (see “Exception Table” on page 112).

The SET INTEGRITY statement does not activate any DELETE triggers as a result of deleting rows that violate constraints, but once the table is removed from check pending state, triggers are active. Thus, if we correct data and insert rows from the exception table into the loaded table, any INSERT triggers defined on the table will be activated. The implications of this should be considered. One option is to drop the INSERT trigger, insert rows from the exception table, and then recreate the INSERT trigger.

Restarting an Interrupted Load Operation

If the load utility cannot start because of a user error, such as a nonexistent data file or invalid column names, it will terminate and leave the table space in a normal state.

If a failure occurs while loading data, you can restart the load operation from the last consistency point (using the RESTART option), or reload the entire table (using the REPLACE option). Specify the same parameters as in the previous invocation, so that the utility can find the necessary temporary files.

Using the Load Copy Location File

The **DB2LOADREC** registry variable is used to identify the file with the load copy location information. This file is used during roll-forward recovery to locate the load copy. It contains information about:

- Media type
- Number of media devices to be used
- Location of the load copy generated during a table load operation
- File name of the load copy, if applicable.

If the location file does not exist, or no matching entry is found in the file, the information from the log record is used.

The information in the file may be overwritten before roll-forward recovery takes place.

In a partitioned database environment, the load copy location file must exist at each database partition server, and the file name (including the path) must be the same.

Following is an example of the location file. The first five parameters must have valid values, and are used to identify the load copy. The entire structure is repeated for each load copy recorded.

```
TIMestamp      19950725182542         * Time stamp generated at load time
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    TORONTO                * DB2INSTANCE
```

```
BUFfernumber   NULL                   * Number of buffers to be used for recovery
SESsionnumber  NULL                   * Number of sessions to be used for recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                        A for ADSM
                                                        O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            O
SHRlib         /@sys/lib/backup_vendor.a
```

Notes:

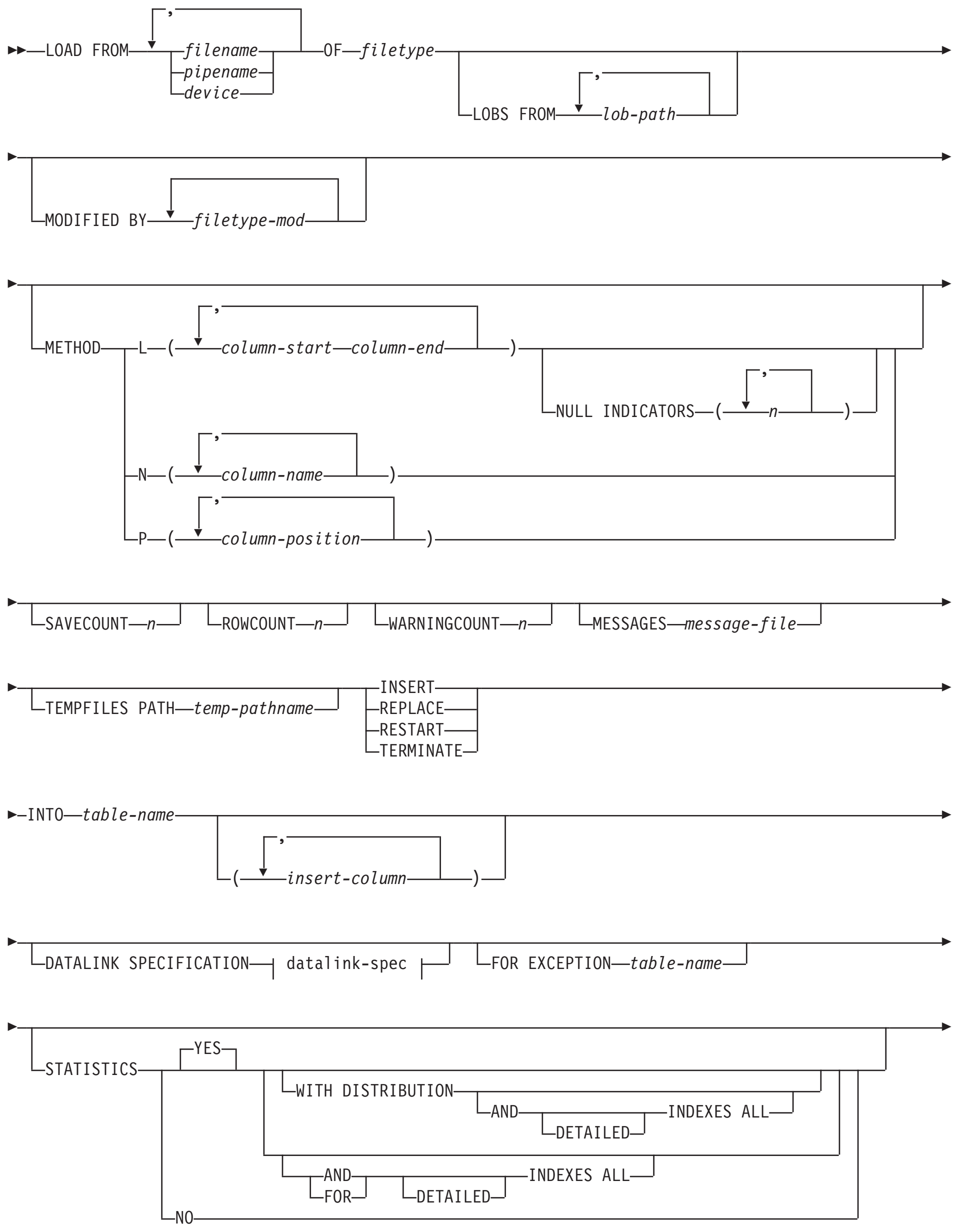
1. The first three characters of each keyword are significant. All keywords must be in the specified order. No blank lines are accepted.
2. The time stamp is in the format *yyyymmddhhmmss*.
3. All fields are mandatory, except for BUF and SES, which can be NULL.
4. The media type can be: local device (L for tape, disk or diskettes), ADSM (A), or other vendor (O). If it is L, the number of locations, followed by the location entries, are required. If the type is A, no further input is required. If the type is O, the shared library name is required. For detailed information about using ADSM and other vendor products as backup media, see the “ADSTAR Distributed Storage Manager” section of the database recovery chapter in the *Administration Guide*.
5. The SHRlib parameter points to a library whose function is to store the LOAD COPY data.

If you run LOAD COPY NO, and do not take a backup copy of the database or affected table spaces after running the load operation, you cannot restore the database or the table spaces to a point in time that is more recent than the load operation. That is, you cannot use roll-forward recovery to rebuild the database or the table spaces to their state after the load operation. You can only restore the database or the table spaces to a point in time that *precedes* the load operation.

If you want to use a particular load copy, the load time stamps are recorded in the recovery history file for the database. In a partitioned database environment, the recovery history file is local to each database partition.

LOAD Command

Command Syntax



```
►──┬──────────────────────────────────────────────────────────────┬──┬──────────────────┬──────────►
   │       ┌─NO─────────────────────────────────────────────────┐  │  └─HOLD QUIESCE─────┘
   ├─COPY──┴─YES──┬─USE ADSM──┬───────────────────────────────┬─┬┴──┤
   │              │           └─OPEN──num-sess──SESSIONS──────┘ │   │
   │              │       ┌─,────────────────┐                  │   │
   │              │       ▼                  │                  │   │
   │              ├─TO────┴─device/directory─┴──────────────────┤   │
   │              └─LOAD──lib-name──┬───────────────────────────┤   │
   │                                └─OPEN──num-sess──SESSIONS──┘   │
   └─NONRECOVERABLE─────────────────────────────────────────────────┘
```

```
►──┬──────────────────────┬──┬──────────────────────────────┬──┬──────────────────────┬──────────►
   └─WITHOUT PROMPTING────┘  └─DATA BUFFER──buffer-size─────┘  └─CPU_PARALLELISM──n───┘
```

```
►──┬────────────────────────┬──┬──────────────────────────────────────┬──────────────────────►◄
   └─DISK_PARALLELISM──n────┘  └─INDEXING MODE──┬─AUTOSELECT───┬──────┘
                                                ├─REBUILD──────┤
                                                ├─INCREMENTAL──┤
                                                └─DEFERRED─────┘
```

datalink-spec:

```
   ┌─,──────────────────────────────────────────────────────────────────────────────────────────┐
   ▼                                                                                            │
├──┴──(──┬──────────────────┬──┬──────────────────────────────────────┬──┬──────────────────────────┬──)──┴──┤
         └─DL_LINKTYPE URL──┘  ├─DL_URL_REPLACE_PREFIX──"prefix"──────┤  └─DL_URL_SUFFIX──"suffix"──┘
                               └─DL_URL_DEFAULT_PREFIX──"prefix"──────┘
```

Command Parameters

COPY NO

Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, *logretain* or *userexit* is on). The data will not be accessible until a table space backup or a full database backup is made.

COPY YES

Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled (both *logretain* and *userexit* are off). The option is not supported for tables with DATALINK columns.

USE ADSM

Specifies that the copy will be stored using ADSTAR Distributed Storage Manager (ADSM).

OPEN num-sess SESSIONS

The number of I/O sessions to be used with ADSM or the vendor product. The default value is 1.

TO device/directory

Specifies the device or directory on which the copy image will be created. Tape is not supported on OS/2; copy to tapes is not supported for DB2 servers running on SCO UnixWare 7.

LOAD lib-name

The name of the shared library (DLL on OS/2 or the Windows operating system) containing the vendor backup and restore I/O functions to be used. It may contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

CPU_PARALLELISM n

Specifies the number of processes or threads that the load utility will spawn for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value at run time.

Notes:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the SAVECOUNT parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When CPU_PARALLELISM is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When CPU_PARALLELISM is set to one, the loader waits on I/O during consistency points. A load operation with CPU_PARALLELISM set to two, and SAVECOUNT set to 10 000, completes faster than the same operation with CPU_PARALLELISM set to one, even though there is only one CPU.

DATA BUFFER buffer-size

Specifies the number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

DATALINK SPECIFICATION

For each DATALINK column, there can be one column specification enclosed by parentheses. Each column specification consists of one or more DL_LINKTYPE, prefix, and a DL_URL_SUFFIX specification. The prefix specification can be either DL_URL_REPLACE_PREFIX or DL_URL_DEFAULT_PREFIX.

There can be as many DATALINK column specifications as the number of DATALINK columns defined in the table. The order of specifications follows the order of DATALINK columns found within the *insert-column* list, or within the table definition (if an *insert-column* list is not specified).

DISK_PARALLELISM n

Specifies the number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

DL_LINKTYPE

If specified, it should match the LINKTYPE of the column definition. Thus, DL_LINKTYPE URL is acceptable if the column definition specifies LINKTYPE URL.

DL_URL_DEFAULT_PREFIX ″prefix″

If specified, it should act as the default prefix for all DATALINK values within the same column. In this context, prefix refers to the ″scheme host port″ part of the URL specification.

Examples of prefix are:

```
"http://server"
"file://server"
"file:"
"http://server:80"
```

If no prefix is found in the column data, and a default prefix is specified with DL_URL_DEFAULT_PREFIX, the default prefix is prefixed to the column value (if not NULL).

For example, if DL_URL_DEFAULT_PREFIX specifies the default prefix "http://toronto":

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://coyote/a/b/c″.
- The column input value NULL is stored as NULL.

DL_URL_REPLACE_PREFIX ″prefix″

This clause is useful when loading or importing data previously generated by the export utility, if the user wants to globally replace the host name in the data with another host name. If specified, it becomes the prefix for *all* non-NULL column values. If a column value has a prefix, this will replace it. If a column value has no prefix, the prefix specified by DL_URL_REPLACE_PREFIX is prefixed to the column value.

For example, if DL_URL_REPLACE_PREFIX specifies the prefix `"http://toronto"`:

- The column input value ″/x/y/z″ is stored as ″http://toronto/x/y/z″.
- The column input value ″http://coyote/a/b/c″ is stored as ″http://toronto/a/b/c″. Note that ″toronto″ replaces ″coyote″.
- The column input value NULL is stored as NULL.

DL_URL_SUFFIX ″suffix″

If specified, it is appended to every non-NULL column value for the column. It is, in fact, appended to the ″path″ component of the data location part of the DATALINK value.

FOR EXCEPTION table-name

Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. DATALINK exceptions are also captured in the exception table.

Information that is written to the exception table is *not* written to the dump file (for a description of the `dumpfile` modifier, see Table 8 on page 104). In a partitioned database environment, an exception table must be defined for those nodes on which the loading table is defined. The dump file, on the other hand, contains rows that cannot be loaded because they are invalid or have syntax errors. For more information, see “Exception Table” on page 112.

FROM filename/pipename/device

Specifies the file, pipe, or device that contains the data being loaded. This file, pipe, or device must reside on the node where the database resides. If several names are specified, they will be processed in sequence. If the last item specified is a tape device, the user is prompted for another tape. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted).

d Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes).

t Terminate. Terminate all devices.

Notes:

1. Tape is not supported on OS/2.
2. It is recommended that the fully qualified file name be used. If the server is remote, the fully qualified file name must be used. If the database resides on the same node as the caller, relative paths may be used.
3. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate.
4. If, when specifying *pipename* on OS/2, less than the expected amount of data is loaded, clean up system resources (IPL is recommended), and reissue the LOAD command.

HOLD QUIESCE

Specifies that the utility should leave the table in quiesced exclusive state after the load operation. To unquiesce the table spaces, issue:

```
db2 quiesce tablespaces for table <tablename> reset
```

Note: Ensure that no *phantom quiesces* are created (see the *Command Reference*).

INDEXING MODE

Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT

The load utility will automatically decide between REBUILD or INCREMENTAL mode.

REBUILD

All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL

Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

Incremental indexing is not supported when all of the following conditions is true:

- The LOAD COPY option is specified (*logretain* or *userexit* is enabled).
- The table resides in a DMS table space.
- The index object resides in a table space that is shared by other table objects belonging to the table being loaded.

To bypass this restriction, it is recommended that indexes be placed in a separate table space.

DEFERRED

The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation may force a rebuild (for more information, see the *Administration Guide*), or indexes may be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

INSERT

One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

insert-column

Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example,

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, Int 4, I5, I6, DT7, I8, TM9)
```

will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

```
db2 load from delfile1 of del modified by noeofchar noheader
   method P (1, 2, 3, 4, 5, 6, 7, 8, 9)
   insert into table1 (BLOB1, S2, I3, "Int 4", I5, I6, DT7, I8, TM9)
```

INTO table-name
Specifies the database table into which the data is to be loaded. This table cannot be a system table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the current authorization ID.

LOBS FROM lob-path
The path to the data files containing LOB values to be loaded. The path must end with a slash (`/`). The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. This option is ignored if `lobsinfile` is not specified within the *filetype-mod* string (see Table 8 on page 104).

MESSAGES message-file
Specifies the destination for warning and error messages that occur during the load operation. If a message file is not specified, messages are written to standard output. If the complete path to the file is not specified, the load utility uses the current directory and the default drive as the destination. If the name of a file that already exists is specified, the utility appends the information.

METHOD

L Specifies the start and end column numbers from which to load data.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each column in the table that is not nullable should be included in this list. Specify only complete subsets of column names (for example, given file columns F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F1,F2,F3,F4) insert into table_name (C1,C2,C3,C4)` is a valid request, while `method N (F1,F4)` is not valid, since there will be no data to put into C3.

Note: This method can only be used with IXF files.

P Specifies the numbers of the columns to be loaded. Each column in the table that is not nullable should be included in this list. Specify only complete subsets of column numbers (for example, given file columns F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT

NULL, and C4 INT, `method P (1,2,3,4)` is a valid request, while `method P (1,4)` is not valid.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY filetype-mod

Specifies additional options (see Table 8 on page 104).

NONRECOVERABLE

Specifies that the load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation.

This option should not be used when DATALINK columns with the FILE LINK CONTROL attribute are present in, or being added to, the table.

NULL INDICATORS n

Specifies a column (by number) to be used as a NULL indicator field. If this option is used, a NULL indicator column for each data column must also be specified. A value of zero indicates that the data column is not nullable, and that there will always be data in that column.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the METHOD L option will be loaded.

The NULL indicator character can be changed using the MODIFIED BY option (see the description of the `nullindchar` modifier in Table 8 on page 104).

OF filetype

Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)
- IXF (integrated exchange format, PC version), exported from the same or from another DB2 table.

For more information about file formats, see “Appendix C. Export/Import/Load Utility File Formats” on page 179.

REPLACE
One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This option is not supported for tables with DATALINK columns.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

RESTARTCOUNT
Reserved.

ROWCOUNT n
Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

SAVECOUNT n
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using "LOAD QUERY Command" on page 82. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

SORT BUFFER buffer-size
Reserved.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

STATISTICS YES
Specifies that statistics are to be collected for the table and for any existing indexes. This option is supported only if the load operation is in REPLACE mode.

WITH DISTRIBUTION
Specifies that distribution statistics are to be collected.

AND INDEXES ALL
Specifies that both table and index statistics are to be collected.

FOR INDEXES ALL
Specifies that only index statistics are to be collected.

DETAILED
Specifies that extended index statistics are to be collected.

TEMPFILES PATH temp-pathname
Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server node.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 4 bytes for each duplicate or rejected row containing DATALINK values
- 136 bytes for each message that the load utility generates
- 15KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the INSERT option is specified, and there is a large amount of long field or LOB data already in the table.

For more information about temporary files, see “Load Temporary Files” on page 113.

TERMINATE
One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a load REPLACE, the table will be truncated to an empty table after the load TERMINATE operation. If the load operation being terminated is a load INSERT, the table will retain all of its original records after the load TERMINATE operation.

If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

USING directory

Reserved.

WARNINGCOUNT n

Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

WITHOUT PROMPTING

Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

If this option is not specified, and the tape device encounters an end of tape for the copy image, or the last item listed is a tape device, the user is prompted for a new tape on that device. Tape is not supported on OS/2.

LOAD QUERY Command

Checks the status of a load operation during processing. A connection to the same database, and a separate CLP session are also required to successfully invoke this command. It can be used either by local or remote users.

Authorization

None

Required Connection

Database

Command Syntax

```
►►──LOAD QUERY──TABLE──table-name──┬──────────────────────────┬──┬──────────────┬──────────►
                                   └─TO──local-message-file──┘  ├─NOSUMMARY────┤
                                                                 └─SUMMARYONLY──┘

►──┬────────────┬───────────────────────────────────────────────────────────────────────►◄
   └─SHOWDELTA──┘
```

Command Parameters

NOSUMMARY
Specifies that no load summary information (rows read, rows skipped, rows loaded, rows rejected, rows deleted, rows committed, and number of warnings) is to be reported.

SHOWDELTA
Specifies that only new information (pertaining to load events that have occurred since the last invocation of the LOAD QUERY command) is to be reported.

SUMMARYONLY
Specifies that only load summary information is to be reported.

TABLE table-name
Specifies the name of the table into which data is currently being loaded.

TO local-message-file
Specifies the destination for warning and error messages that occur during the load operation. This file cannot be the *message-file* specified for the LOAD command. If the file already exists, all messages that the load utility has generated are appended to it.

Examples

A user loading a large amount of data into the STAFF table wants to check the status of the load operation. The user can specify:

```
   db2 connect to <database>
   db2 load query table staff to /u/mydir/staff.tempmsg
```

The output file /u/mydir/staff.tempmsg might look like the following:

```
SQL3500W The utility is beginning the "LOAD" phase at time
"02-13-1997 19:40:29.645353".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3109N The utility is beginning to load data from file
"/u/mydir/data/staffbig.ixf".

SQL3150N The H record in the PC/IXF file has product "DB2 01.00",
date "19970111", and time "194554".

SQL3153N The T record in the PC/IXF file has name
"data/staffbig.ixf", qualifier " ", and source " ".

SQL3519W  Begin Load Consistency Point. Input record count =
"111152".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count =
"222304".

SQL3520W  Load Consistency Point was successful.
```

See Also

"LOAD Command" on page 70.

Load API

C API Syntax

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqluload (
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

Generic API Syntax

```
/* File: sqlutil.h */
/* API: Load */
/* ... */
SQL_API_RC SQL_API_FN
  sqlgload (
    unsigned short FileTypeLen,
    unsigned short LocalMsgFileNameLen,
    unsigned short RemoteMsgFileNameLen,
    sqlu_media_list * pDataFileList,
    sqlu_media_list * pLobPathList,
    struct sqldcol * pDataDescriptor,
    struct sqlchar * pActionString,
    char * pFileType,
    struct sqlchar * pFileTypeMod,
    char * pLocalMsgFileName,
    char * pRemoteMsgFileName,
    short CallerAction,
    struct sqluload_in * pLoadInfoIn,
    struct sqluload_out * pLoadInfoOut,
    sqlu_media_list * pWorkDirectoryList,
    sqlu_media_list * pCopyTargetList,
    long * pNullIndicators,
    void * pReserved,
    struct sqlca * pSqlca);
/* ... */
```

API Parameters

FileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the file type.

LocalMsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the local message file name.

RemoteMsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the temporary files path name.

pDataFileList
Input. A pointer to an *sqlu_media_list* structure used to provide a list of source files, devices, vendors or pipes. Tape is not supported on OS/2.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in sqlutil) are:

SQLU_SERVER_LOCATION
If the *media_type* field is set to this value, the caller provides information through *sqlu_location_entry* structures. The *sessions*

field indicates the number of *sqlu_location_entry* structures provided. This is used for files, devices, and named pipes.

SQLU_ADSM_MEDIA
If the *media_type* field is set to this value, the *sqlu_vendor* structure is used, where *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of ADSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

SQLU_OTHER_MEDIA
If the *media_type* field is set to this value, the *sqlu_vendor* structure is used, where *shr_lib* is the shared library name, and *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

Wherever a file name is provided, it should be fully qualified. For more information, see “SQLU-MEDIA-LIST ” in the *Administrative API Reference*.

pLobPathList
Input. A pointer to an *sqlu_media_list* structure. For IXF, ASC, and DEL file types, a list of fully qualified paths or devices to identify the location of the individual LOB files to be loaded. The file names are found in the IXF, ASC, or DEL files, and are appended to the paths provided. Tape is not supported on OS/2.

The information provided in this structure depends on the value of the *media_type* field. Valid values (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information through *sqlu_media_entry* structures. The *sessions* field indicates the number of *sqlu_media_entry* structures provided.

SQLU_ADSM_MEDIA
If set to this value, the *sqlu_vendor* structure is used, where *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of ADSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

SQLU_OTHER_MEDIA

If set to this value, the *sqlu_vendor* structure is used, where *shr_lib* is the shared library name, and *filename* is the unique identifier for the data to be loaded. There should only be one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one *sqlu_vendor* entry.

For more information, see "SQLU-MEDIA-LIST " in the *Administrative API Reference*.

pDataDescriptor

Input. Pointer to an *sqldcol* structure containing information about the columns being selected for loading from the external file.

If the *pFileType* parameter is set to `SQL_ASC`, the *dcolmeth* field of this structure must be set to `SQL_METH_L`. The user specifies the start and end locations for each column to be loaded.

If the file type is `SQL_DEL`, *dcolmeth* can be either `SQL_METH_P` or `SQL_METH_D`. If it is `SQL_METH_P`, the user must provide the source column position. If it is `SQL_METH_D`, the first column in the file is loaded into the first column of the table, and so on.

If the file type is `SQL_IXF`, *dcolmeth* can be one of `SQL_METH_P`, `SQL_METH_D`, or `SQL_METH_N`. The rules for DEL files apply here, except that `SQL_METH_N` indicates that file column names are to be provided in the *sqldcol* structure.

For more information, see "SQLDCOL" in the *Administrative API Reference*.

pActionString

Input. Pointer to an *sqlchar* structure containing a 2-byte long field, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[FOR EXCEPTION e_tbname]"
```

INSERT

Adds the loaded data to the table without changing the existing table data.

REPLACE
Deletes all existing data from the table, and inserts the loaded data. The table definition and the index definitions are not changed.

RESTART
Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

tbname The name of the table into which the data is to be loaded. The table cannot be a system table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the current authorization ID.

(*column_list***)**
A list of table column names into which the data is to be inserted. The column names must be separated by commas. If a name contains spaces or lowercase characters, it must be enclosed by quotation marks.

DATALINK SPECIFICATION *datalink-spec*
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the LOAD command (see “LOAD Command” on page 70).

FOR EXCEPTION *e_tbname*
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. DATALINK exceptions are also captured in the exception table.

pFileType

Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in `sqlutil`) are:

SQL_ASC

Non-delimited ASCII.

SQL_DEL

Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF

PC version of the Integrated Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

For more information about file formats, see “Appendix C. Export/Import/Load Utility File Formats” on page 179.

pFileTypeMod

Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types.

For more information, see “SQLCHAR” in the *Administrative API Reference*, and “File Type Modifiers (Load)” on page 104.

pLocalMsgFileName

Input. A string containing the name of a local file to which output messages are to be written.

pRemoteMsgFileName

Input. A string containing the path name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and delete phase information. For more information about temporary files, see “Load Temporary Files” on page 113.

CallerAction

Input. An action requested by the caller. Valid values (defined in `sqlutil`) are:

SQLU_INITIAL

Initial call. This value (or SQLU_NOINTERRUPT) must be used on the first call to the API.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. This value (or SQLU_INITIAL) must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested load operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_ABORT
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_RESTART
Restart processing.

SQLU_DEVICE_TERMINATE
Terminate a single device. This option should be specified if the utility is to stop reading data from the device, but further processing of the data is to be done.

pLoadInfoIn
Input. Optional pointer to the *sqluload_in* structure containing additional input parameters. For information about this structure, see “Data Structure: SQLULOAD-IN” on page 93.

pLoadInfoOut
Output. Optional pointer to the *sqluload_out* structure containing additional output parameters. For information about this structure, see “Data Structure: SQLULOAD-OUT” on page 97.

pWorkDirectoryList
Reserved.

pCopyTargetList

Input. A pointer to an *sqlu_media_list* structure used (if a copy image is to be created) to provide a list of target paths, devices, or a shared library to which the copy image is to be written.

The values provided in this structure depend on the value of the *media_type* field. Valid values for this field (defined in `sqlutil`) are:

SQLU_LOCAL_MEDIA

If the copy is to be written to local media, set the *media_type* to this value and provide information about the targets in *sqlu_media_entry* structures. The *sessions* field specifies the number of *sqlu_media_entry* structures provided.

SQLU_ADSM_MEDIA

If the copy is to be written to ADSM, use this value. No further information is required.

SQLU_OTHER_MEDIA

If a vendor product is to be used, use this value and provide further information via an *sqlu_vendor* structure. Set the *shr_lib* field of this structure to the shared library name of the vendor product. Provide only one *sqlu_vendor* entry, regardless of the value of *sessions*. The *sessions* field specifies the number of *sqlu_media_entry* structures provided. The load utility will start the sessions with different sequence numbers, but with the same data provided in the one *sqlu_vendor* entry.

For more information, see "SQLU-MEDIA-LIST " in the *Administrative API Reference*.

pNullIndicators

Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the data file. That is, the number of elements must equal the *dcolnum* field of the *pDataDescriptor* parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a NULL indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a `Y` or an `N`. A `Y` indicates that the table column data is NULL, and `N` indicates that the table column data is not NULL.

pReserved

Reserved for future use.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* and the *SQL Reference*.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See the *Application Development Guide*. For a description of the syntax, see “LOAD Command” on page 70.

Data Structure: SQLULOAD-IN

This structure is used to input information during a call to “Load API” on page 84.

Table 6. Fields in the SQLULOAD-IN Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	UNSIGNED LONG	Size of this structure in bytes.
SAVECNT	UNSIGNED LONG	The number of records to load before establishing a consistency point. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using “db2LoadQuery - Load Query API” on page 99. If the value of *savecnt* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance. The default value is 0, meaning that no consistency points will be established, unless necessary.
RESTARTCNT	UNSIGNED LONG	Reserved.
ROWCNT	UNSIGNED LONG	The number of physical records to be loaded. Allows a user to load only the first *rowcnt* rows in a file.
WARNINGCNT	UNSIGNED LONG	Stops the load operation after *warningcnt* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If *warningcnt* is 0, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was exceeded, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

Table 6. Fields in the SQLULOAD-IN Structure (continued)

Field Name	Data Type	Description
DATA_BUFFER_SIZE	UNSIGNED LONG	The number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the required minimum is used, and no warning is returned. This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter. If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.
SORT_BUFFER_SIZE	UNSIGNED LONG	Reserved.
HOLD_QUIESCE	UNSIGNED SHORT	A flag whose value is set to TRUE if the utility is to leave the table in quiesced exclusive state after the load, and to FALSE if it is not.
RESTARTPHASE	CHAR(1)	Reserved.
STATSOPT	CHAR(1)	Granularity of statistics to collect. See below for values.
CPU_PARALLELISM	UNSIGNED SHORT	The number of processes or threads that the load utility will spawn for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, the load utility uses an intelligent default value at run time. **Note:** If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs, or the value specified by the user.
DISK_PARALLELISM	UNSIGNED SHORT	The number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

Table 6. Fields in the SQLULOAD-IN Structure (continued)

Field Name	Data Type	Description
NON_RECOVERABLE	UNSIGNED SHORT	Set to SQLU_NON_RECOVERABLE_LOAD if the load transaction is to be marked as non-recoverable, and it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as ″invalid″. The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped. With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. Set to SQLU_RECOVERABLE_LOAD if the load transaction is to be marked as recoverable.

Valid values for *STATSOPT* (defined in sqlutil) are:

SQLU_STATS_NONE

SQL_STATS_EXTTABLE_ONLY

SQL_STATS_EXTTABLE_INDEX

SQL_STATS_INDEX

SQL_STATS_TABLE

SQL_STATS_EXTINDEX_ONLY

SQL_STATS_EXTINDEX_TABLE

SQL_STATS_ALL

SQL_STATS_BOTH

Data Structure: SQLULOAD-IN

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-IN */
/* ... */
SQL_STRUCTURE sqluload_in
{
  unsigned long   sizeOfStruct;
  unsigned long   savecnt;
  unsigned long   restartcnt;
  unsigned long   rowcnt;
  unsigned long   warningcnt;
  unsigned long   data_buffer_size;
  unsigned long   sort_buffer_size;  /* No longer used. */
  unsigned short  hold_quiesce;
  char            restartphase;
  char            statsopt;
  unsigned short  cpu_parallelism;
  unsigned short  disk_parallelism;
  unsigned short  non_recoverable;
};
/* ... */
```

COBOL Structure

```
* File: sqlutil.cbl
01 SQLULOAD-IN.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 40.
    05 SQL-SAVECNT            PIC 9(9) COMP-5.
    05 SQL-RESTARTCOUNT       PIC 9(9) COMP-5.
    05 SQL-ROWCNT             PIC 9(9) COMP-5.
    05 SQL-WARNINGCNT         PIC 9(9) COMP-5.
    05 SQL-DATA-BUFFER-SIZE   PIC 9(9) COMP-5.
    05 SQL-SORT-BUFFER-SIZE   PIC 9(9) COMP-5.  * No longer used.
    05 SQL-HOLD-QUIESCE       PIC 9(4) COMP-5.
    05 SQL-RESTARTPHASE       PIC X.
    05 SQL-STATSOPT           PIC X.
    05 SQL-CPU-PARALLELISM    PIC 9(4) COMP-5.
    05 SQL-DISK-PARALLELISM   PIC 9(4) COMP-5.
    05 SQL-NON-RECOVERABLE    PIC 9(4) COMP-5.
    05 FILLER                 PIC X(2).
*
```

Data Structure: SQLULOAD-OUT

This structure is used to output information after a call to “Load API” on page 84.

Table 7. Fields in the SQLULOAD-OUT Structure

Field Name	Data Type	Description
SIZEOFSTRUCT	UNSIGNED LONG	Size of this structure in bytes.
ROWSREAD	UNSIGNED LONG	Number of records read during the load operation.
ROWSSKIPPED	UNSIGNED LONG	Number of records skipped before the load operation begins.
ROWSLOADED	UNSIGNED LONG	Number of rows loaded into the target table.
ROWSREJECTED	UNSIGNED LONG	Number of records that could not be loaded.
ROWSDELETED	UNSIGNED LONG	Number of duplicate rows deleted.
ROWSCOMMITTED	UNSIGNED LONG	The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

Language Syntax

C Structure

```
/* File: sqlutil.h */
/* Structure: SQLULOAD-OUT */
/* ... */
SQL_STRUCTURE sqluload_out
{
  unsigned long   sizeOfStruct;
  unsigned long   rowsRead;
  unsigned long   rowsSkipped;
  unsigned long   rowsLoaded;
  unsigned long   rowsRejected;
  unsigned long   rowsDeleted;
  unsigned long   rowsCommitted;
};
/* ... */
```

Data Structure: SQLULOAD-OUT

COBOL Structure

```
* File: sqlutil.cbl
01 SQLULOAD-OUT.
    05 SQL-SIZE-OF-STRUCT     PIC 9(9) COMP-5 VALUE 28.
    05 SQL-ROWS-READ          PIC 9(9) COMP-5.
    05 SQL-ROWS-SKIPPED       PIC 9(9) COMP-5.
    05 SQL-ROWS-LOADED        PIC 9(9) COMP-5.
    05 SQL-ROWS-REJECTED      PIC 9(9) COMP-5.
    05 SQL-ROWS-DELETED       PIC 9(9) COMP-5.
    05 SQL-ROWS-COMMITTED     PIC 9(9) COMP-5.
*
```

db2LoadQuery - Load Query API

Checks the status of a load operation during processing.

Authorization

None

Required Connection

Database

API Include File

db2ApiDf.h

C API Syntax

```
/* File: db2ApiDf.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
db2LoadQuery (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  char * piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct * poOutputStruct;
  char * piLocalMessageFile;
} db2LoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
} db2LoadQueryOutputStruct;
/* ... */
```

Generic API Syntax

```
/* File: db2ApiDf.h */
/* API: Load Query */
/* ... */
SQL_API_RC SQL_API_FN
db2GenLoadQuery (
  db2Uint32 versionNumber,
  void * pParmStruct,
  struct sqlca *pSqlca);

typedef struct
{
  db2Uint32 iStringType;
  db2Uint32 iStringLen;
  char * piString;
  db2Uint32 iShowLoadMessages;
  db2LoadQueryOutputStruct * poOutputStruct;
  db2Uint32 iLocalMessageFileLen;
  char * piLocalMessageFile
} db2LoadQueryStruct;

typedef struct
{
  db2Uint32 oRowsRead;
  db2Uint32 oRowsSkipped;
  db2Uint32 oRowsCommitted;
  db2Uint32 oRowsLoaded;
  db2Uint32 oRowsRejected;
  db2Uint32 oRowsDeleted;
  db2Uint32 oCurrentIndex;
  db2Uint32 oNumTotalIndexes;
  db2Uint32 oCurrentMPPNode;
  db2Uint32 oLoadRestarted;
  db2Uint32 oWhichPhase;
  db2Uint32 oWarningCount;
} db2LoadQueryOutputStruct;
/* ... */
```

API Parameters

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter, *pParmStruct*.

pParmStruct
Input. A pointer to the *db2LoadQueryStruct* structure.

pSqlca
Output. A pointer to the *sqlca* structure. For more information about this structure, see the *Administrative API Reference* and the *SQL Reference*.

iStringType
Input. Specifies a type for *piString*. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_TABLENAME
Represents specifying a table name for use by the **db2LoadQuery** API.

iStringLen
Input. Specifies the length in bytes of *piString*.

piString
Input. Specifies a temporary files path name or a table name, depending on the value of *iStringType*.

iShowLoadMessages
Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_SHOW_ALL_MSGS
Return all load messages.

DB2LOADQUERY_SHOW_NO_MSGS
Return no load messages.

DB2LOADQUERY_SHOW_NEW_MSGS
Return only messages that have been generated since the last call to this API.

poOutputStruct
Output. A pointer to the *db2LoadQueryOutputStruct* structure, which contains load summary information. Set to NULL if a summary is not required.

iLocalMessageFileLen
Input. Specifies the length in bytes of *piLocalMessageFile*.

piLocalMessageFile
Input. Specifies the name of a local file to be used for output messages.

oRowsRead
Output. Number of records read so far by the load utility.

oRowsSkipped
Output. Number of records skipped before the load operation began.

oRowsCommitted
Output. Number of rows committed to the target table so far.

oRowsLoaded
Output. Number of rows loaded into the target table so far.

oRowsRejected
Output. Number of rows rejected from the target table so far.

oRowsDeleted
Output. Number of rows deleted from the target table so far (during the delete phase).

oCurrentIndex
Output. Index currently being built (during the build phase).

oCurrentMPPNode
Output. Indicates which node is being queried (for MPP mode only).

oLoadRestarted
Output. A flag whose value is `TRUE` if the load operation being queried is a load restart operation.

oWhichPhase
Output. Indicates the current phase of the load operation being queried. Valid values (defined in `db2ApiDf.h`) are:

DB2LOADQUERY_LOAD_PHASE
Load phase.

DB2LOADQUERY_BUILD_PHASE
Build phase.

DB2LOADQUERY_DELETE_PHASE
Delete phase.

oNumTotalIndexes
Output. Total number of indexes to be built (during the build phase).

oWarningCount
Output. Total number of warnings returned so far.

REXX API Syntax

This API can be called from REXX through the SQLDB2 interface. See the *Application Development Guide*. For a description of the syntax, see “LOAD QUERY Command” on page 82.

Sample Programs

C	\sqllib\samples\c\loadqry.sqc
COBOL	\sqllib\samples\cobol\loadqry.sqb
FORTRAN	\sqllib\samples\fortran\loadqry.sqf

Usage Notes

This API reads the status of the load operation on the table specified by *piString*, and writes the status to the file specified by *pLocalMsgFileName*.

File Type Modifiers (Load)

Table 8. Valid File Type Modifiers (LOAD)

Modifier	Description
All File Formats	
anyorder	This modifier is used in conjunction with the *cpu_parallelism* parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of *cpu_parallelism* is 1, this option is ignored. This option is not supported if SAVECOUNT > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly. For example, if a value of `123qwr4` were to be encountered as a field entry for an integer column in an ASC file, the load utility would ordinarily flag a syntax error, since the value does not represent a valid number. With `fastparse`, a syntax error is not detected, and an arbitrary number is loaded into the integer field. Care must be taken to use this modifier with clean data only. Performance improvements using this option with ASCII data can be quite substantial, but `fastparse` does not significantly enhance performance with PC/IXF data, since IXF is a binary format, and `fastparse` affects parsing and conversion from ASCII to internal forms.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when loading the index. The first entry in a page is added without restriction; subsequent entries are added if the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement, and affects index leaf pages only.
lobsinfile	*lob-path* specifies the path to the files containing LOB values. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**
noheader	Skips the header verification code. The AutoLoader utility (see “Chapter 4. AutoLoader” on page 131) writes a header to each file contributing data to a table in a multi-node nodegroup. The header includes the node number, the partitioning map, and the partitioning key specification. The load utility requires this information to verify that the data is being loaded at the correct node. When loading files into a table that exists on a single-node nodegroup, the headers do not exist, and this option causes the load utility to skip the header verification code.
norowwarnings	Suppresses all warnings about rejected rows.
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. **Note:** The PCTFREE value of a table determines the amount of free space designated per page. If a `pagefreespace` value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by `pagefreespace` overrides the PCTFREE value specified for the table.
totalfreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages, 20 additional empty pages will be appended. The total number of data pages for the table will be 120.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: ",," is specified for the column • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. Without this option, if a source column contains no data for a row instance, one of the following occurs: • If the column is nullable, a NULL is loaded • If the column is not nullable, the utility rejects the row.
ASCII File Formats (ASC/DEL)	
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters may not coincide with the shift-in and shift-out DBCS characters. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description

dumpfile = *x*	*x* is the fully qualified (according to the server node) name of an exception file to which rejected rows are written. A maximum of 32KB of data is written per record. Following is an example that shows how to specify a dump file:
	`db2 load from data of del`
	`   modified by dumpfile = /u/user/filename`
	`   insert into table_name`
	Notes:
	1. In a partitioned database environment, the path should be local to the loading node, so that concurrently running load operations do not attempt to write to the same file.
	2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass.
	3. This modifier does not support file names with multiple file extensions. For example,
	`dumpfile = /home/svtdbm6/DUMP.FILE`
	is acceptable to the load utility, but
	`dumpfile = /home/svtdbm6/DUMP.LOAD.FILE`
	is not.

implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`.
noeofchar	The optional end-of-file character `x'1A'` is not recognized as the end of file. Processing continues as if it were a normal character.
ASC (Non-delimited ASCII) File Format	

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The `noeofchar` option is assumed. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. **Note:** NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[b] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	**Description**

packeddecimal	Loads packed-decimal data directly, since the `binarynumerics` modifier does not include the DECIMAL field type.
	This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The `noeofchar` option is assumed.
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	Note: NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on OS/2 or on the Windows operating system, the byte order must not be reversed.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. **Note:** This option replaces the obsolete `t` option, which is supported for back-level compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. **Note:** This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.
DEL (Delimited ASCII) File Format	

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[ab] The single quotation mark (') can also be specified as a character string delimiter as follows: `modified by chardel''`
coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[ab]
datesiso	Date format. Causes all date data values to be loaded in ISO format.
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[ab]

delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax:
	`db2 load ... modified by delprioritychar ...`
	For example, given the following DEL data file:
	`"Smith, Joshua",4000,34.98<row delimiter>`
	`"Vincent,<row delimiter>, is a manager", ...`
	`... 4005,44.37<row delimiter>`
	With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.

Table 8. Valid File Type Modifiers (LOAD) (continued)

Modifier	Description
dldel*x*	*x* is a single character DATALINK delimiter. The default value is a semicolon (;). The specified character is used in place of a semicolon as the inter-field separator for a DATALINK value. It is needed because a DATALINK value may have more than one sub-value. [abc] **Note:** *x* must not be the same character specified as the row, column, or character string delimiter.
nodoubledel	Suppresses recognition of double character delimiters.
IXF File Format	
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Notes:

1. The load utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the load operation fails, and an error code is returned.
2. [a] “Delimiter Restrictions” on page 16 lists restrictions that apply to the characters that can be used as delimiter overrides.
3. [b] The character must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
4. [c] Even if the DATALINK delimiter character is a valid character within the URL syntax, it will lose its special meaning within the scope of the load operation.

Exception Table

The exception table is a user-created table that reflects the definition of the table being loaded, and includes some additional columns. It is specified by the FOR EXCEPTION clause on the LOAD command. The table is used to store copies of rows that violate unique index rules; the utility will not check for constraints or foreign key violations other than violations of uniqueness. DATALINK exceptions are also captured in the exception table.

A unique key is a key for which no two values are equal. The mechanism used to enforce this constraint is called a unique index. A primary key is a special case of a unique key. A table cannot have more than one primary key.

Note: Any rows rejected because of invalid data before the building of an index are not inserted into the exception table.

Rows are appended to existing information in the exception table; this can include invalid rows from previous load operations. If you want only the invalid rows from the current load operation, you must remove the existing rows before invoking the utility.

The exception table used with the load utility is identical to the exception tables used by the SET INTEGRITY statement.

An exception table should be used when loading data which has a unique index and the possibility of duplicate records. If an exception table is not specified, and duplicate records are found, the load operation continues, and only a warning message is issued about the deleted duplicate records. The records themselves are not logged.

After the load operation completes, information in the exception table can be used to correct data that is in error. The corrected data can then be inserted into the table.

For more information about exception tables, see the *SQL Reference*.

Dump File

Specifying the *dumpfile* modifier tells the load utility the name and the location of the exception file to which rejected rows are written. When running in a partitioned database environment, the name is given an extension that identifies the partition number where the exceptions were generated. For example:

```
dumpfile = "/u/usrname/dumpit"
```

On partition zero, this will generate a file named /u/usrname/dumpit.000. On partition five, it will generate a file named /u/usrname/dumpit.005, and so on.

Only the first 32 768 bytes of a record are written into the dump file; the rest is discarded.

For more information about load file type modifiers, see “File Type Modifiers (Load)” on page 104.

Load Temporary Files

DB2 creates temporary binary files during load processing. These files are removed when the load operation completes without error.

The temporary files are written to a path that can be specified through the *temp-pathname* parameter of the LOAD command, or in the *pRemoteMsgFileName* parameter of the **sqluload** API. The default path is a subdirectory of the database directory.

The temporary files path resides on the server machine and is accessed by the DB2 instance exclusively. Therefore, it is imperative that any path name qualification given to the *temp-pathname* parameter reflects the directory structure of the server, not the client, and that the DB2 instance owner has read and write permission on the path.

Note: In an MPP system, the temporary files path must reside on a local disk, not on an NFS mount. If the path is on an NFS mount, there will be a significant performance decrement during the load operation.

Attention: The temporary files written to this path must not be tampered with under any circumstances. Doing so will cause the load operation to malfunction, and will place your database in jeopardy.

Load Utility Log Records

The utility manager produces log records associated with a number of DB2 utilities, including the load utility. The log records mark the beginning or the end of a specific activity. The following log records are associated with load operations:

- Load Start. This log record is associated with the beginning of a load operation.

- Table Load Delete Start. This log record is associated with the beginning of the delete phase in a load operation. The delete phase is started only if there are duplicate primary key values.
- Load Delete Start Compensation. This log record is associated with the end of the delete phase in a load operation.
- Load Pending List. This log record is written when a load transaction commits. The pending list is a linked list of non-recoverable operations that are deferred until the transaction commits. No commit log record follows this transaction.

For a description of the structure of these log records, see the *Administrative API Reference*.

Character Set and NLS Considerations

Unequal code page situations, involving expansion or contraction of the character data, can sometimes occur. For example, Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) may have different length encodings for the same character. Normally, comparison of input data length to target column length is performed before reading in any data. If the input length is greater than the target length, NULLs are inserted into that column if it is nullable. Otherwise, the request is rejected. If the `nochecklengths` modifier (see "File Type Modifiers (Load)" on page 104) is specified, no initial comparison is performed, and an attempt is made to load the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is loaded.

Example Load Sessions

CLP Examples

Example 1

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20

- ELE2 positions 21 to 22
- ELE5 positions 23 to 23
- ELE3 positions 24 to 27
- ELE4 positions 28 to 31
- ELE6 positions 32 to 32
- ELE6 positions 33 to 40

Data Records:

```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

Notes:

1. The specification of `striptblanks` in the MODIFIED BY parameter forces the truncation of blanks in VARCHAR columns (COL1, for example, which is 11, 17 and 19 bytes long, in rows 1, 2 and 3, respectively).
2. The specification of `reclen=40` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not used to load the table.
3. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
4. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(........)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
5. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
6. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Example 2 (Loading LOBs from Files)

TABLE1 has 3 columns:

- COL1 CHAR 4 NOT NULL WITH DEFAULT
- LOB1 LOB
- LOB2 LOB

ASCFILE1 has 3 elements:

- ELE1 positions 01 to 04
- ELE2 positions 06 to 13
- ELE3 positions 15 to 22

The following files reside in either `/u/user1` or `/u/user1/bin`:

- ASCFILE2 has LOB data
- ASCFILE3 has LOB data
- ASCFILE4 has LOB data
- ASCFILE5 has LOB data
- ASCFILE6 has LOB data
- ASCFILE7 has LOB data

Data Records in ASCFILE1:

```
1...5....10...15...20...25...30.
REC1 ASCFILE2 ASCFILE3
REC2 ASCFILE4 ASCFILE5
REC3 ASCFILE6 ASCFILE7
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc
   lobs from /u/user1, /u/user1/bin
   modified by lobsinfile reclen=22
   method L (1 4, 6 13, 15 22)
   insert into table1
```

Notes:

1. The specification of `lobsinfile` in the MODIFIED BY parameter tells the loader that all LOB data is to be loaded from files.
2. The specification of `reclen=22` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 22 bytes long.
3. LOB data is contained in 6 files, ASCFILE2 through ASCFILE7. Each file contains the data that will be used to load a LOB column for a specific row. The relationship between LOBs and other data is specified in ASCFILE1. The first record of this file tells the loader to place REC1 in COL1 of row 1. The contents of ASCFILE2 will be used to load LOB1 of

row 1, and the contents of ASCFILE3 will be used to load LOB2 of row 1. Similarly, ASCFILE4 and ASCFILE5 will be used to load LOB1 and LOB2 of row 2, and ASCFILE6 and ASCFILE7 will be used to load the LOBs of row 3.

4. The LOBS FROM parameter contains 2 paths that will be searched for the named LOB files when those files are required by the loader.
5. To load LOBs directly from ASCFILE1 (a non-delimited ASCII file), without the `lobsinfile` modifier, the following rules must be observed:
   - The total length of any record, including LOBs, cannot exceed 32KB.
   - LOB fields in the input records must be of fixed length, and LOB data padded with blanks as necessary.
   - The `striptblanks` modifier must be specified, so that the trailing blanks used to pad LOBs can be removed as the LOBs are inserted into the database.

Example 3 (Using Dump Files)

Table FRIENDS is defined as:

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

If an attempt is made to load the following data records into this table,

```
23, 24, bobby
, 45, john
4,, mary
```

the second row is rejected because the first INT is NULL, and the column definition specifies NOT NULL. Columns which contain initial characters that are not consistent with the DEL format will generate an error, and the record will be rejected. Such records can be written to a dump file (see Table 8 on page 104).

DEL data appearing in a column outside of character delimiters is ignored, but does generate a warning. For example:

```
22,34,"bob"
24,55,"sam" sdf
```

The utility will load ″sam″ in the third column of the table, and the characters ″sdf″ will be flagged in a warning. The record is not rejected. Another example:

```
22 3, 34,"bob"
```

The utility will load `22,34,"bob"`, and generate a warning that some data in column one following the 22 was ignored. The record is not rejected.

Example 4 (Loading DATALINK Data)

The following command loads the table MOVIETABLE from the input file `delfile1`, which has data in the DEL format:

```
db2 load from delfile1 of del
    modified by dldel|
    insert into movietable (actorname, description, url_making_of, url_movie)
    datalink specification (dl_url_default_prefix "http://narang"),
    (dl_url_replace_prefix "http://bomdel" dl_url_suffix ".mpeg")
    for exception excptab
```

Notes:

1. The table has four columns:

actorname	VARCHAR(n)
description	VARCHAR(m)
url_making_of	DATALINK (with LINKTYPE URL)
url_movie	DATALINK (with LINKTYPE URL)

2. The DATALINK data in the input file has the vertical bar (|) character as the sub-field delimiter.
3. If any column value for url_making_of does not have the prefix character sequence, ″http://narang″ is used.
4. Each non-NULL column value for url_movie will get ″http://bomdel″ as its prefix. Existing values are replaced.
5. Each non-NULL column value for url_movie will get ″.mpeg″ appended to the path. For example, if a column value of url_movie is ″http://server1/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″; if the value is ″/x/y/z″, it will be stored as ″http://bomdel/x/y/z.mpeg″.
6. If any unique index or DATALINK exception occurs while loading the table, the affected records are deleted from the table and put into the exception table `excptab`.

API Examples

The following sample program shows how to:

- Export information from the EMP_RESUME table in the SAMPLE database to the file `EXPTABLE.DEL`.
- Load that information from the delimited text file to a new table, LOADTABLE.

For detailed information about the SAMPLE database, see the *Administration Guide*.

The source file for this sample program (`tload.sqc`) can be found in the `\sqllib\samples\c` directory. It contains both DB2 APIs and embedded SQL calls. The script file `bldvaemb.cmd`, located in the same directory, contains the commands to build this and other sample programs. For general information about creating applications containing DB2 administrative APIs, and detailed information about compile and link options, see the *Application Building Guide*.

To build the sample program tload from the source file tload.sqc on OS/2:

1. Copy the files tload.sqc, bldvaemb.cmd, util.c, and util.h to a working directory.
2. If the database manager is not running, issue the command db2start.
3. Enter bldvaemb tload sample. The following files are generated:

```
tload.bnd
tload.c
util.obj
tload.obj
tload.exe
```

To run the sample program (executable file), enter tload. You might find it useful to examine some of the generated files, such as the message files, and the delimited ASCII data file.

```
/*******************************************************************************
**
** Source File Name = tload.sqc  1.4
**
** Licensed Materials - Property of IBM
**
** (C) COPYRIGHT International Business Machines Corp. 1995, 1997
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
**
**
**    PURPOSE :
**      To show the use of the QUIESCE TABLESPACE and the LOAD APIs.
**        - EXPORT the EMP_RESUME table into a comma delimited file.
**        - create a temporary table ('loadtable').
**        - QUIESCE the TABLESPACES, preparing the temporary table to be
**          LOADable.
**        - LOAD the comma delimited file into a temporary table ('loadtable').
**
**    STRUCTURES USED :
**       sqldcol
**       sqlchar
**       sqluexpt_out
**       sqlca
**
**    APIs USED :
**       EXPORT                           sqluexpr
**       QUIESCE TABLESPACE FOR TABLES    sqluvqdp
**       LOAD                             sqluload
**
**    FUNCTIONS DECLARED :
```

```
**       'C' COMPILER LIBRARY :
**          stdio.h  -  printf
**          string.h -  fgets, strncpy
**
**       DBMS LIBRARY :
**          sqlenv.h -  see "APIs USED" above
**
**       OTHER :
**          external :
**             check_error :     Checks for SQLCODE error, and prints out any
**             [in UTIL.C]       related information available.
**
**    EXTERNAL DEPENDANCIES :
**       - Ensure existence of database (SAMPLE) for precompile purposes.
**       - Precompile with the SQL precompiler (PREP in DB2)
**       - Bind to a database (BIND in DB2)
**       - Compile and link with the IBM Cset++ compiler (AIX and OS/2)
**         or the Microsoft Visual C++ compiler (Windows)
**         or the compiler supported on your platform.
**
*******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <malloc.h>

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;
#ifdef DB2AIX
#define  WORKDIR  "/u/workdir"
#else
#define WORKDIR "."
#endif

EXEC SQL INCLUDE SQLCA;
int main (int argc, char *argv[]) {
   short int            callerAction = 0;
   struct sqldcol       DataDescriptor;
   struct sqlchar       *ActionString;
   struct sqlchar       *FileTypeMod;
   struct sqluexpt_out  outputInfo;
   char        datafile[] = "EXPTABLE.DEL";
   char        statement[] = "SELECT empno, photo_format, picture FROM emp_photo";
   char        impStatement[] = "INSERT INTO loadtable (num, format, photo)";
/*
   char        statement[] = "SELECT empno, photo_format FROM emp_photo";
   char        impStatement[] = "INSERT INTO loadtable (num, format)";
*/

   char        msgfile_x[] = "EXPMSG.TXT";
   char        FileType[] = SQL_DEL;


   char        table_name[18];
```

```
 /* Variables for the LOAD API */
 struct sqlu_media_list       DataFileList;
 struct sqlu_media_list       *pLobPathList;
 struct sqluload_in           InputInfo;
 struct sqluload_out          OutputInfo;
 struct sqlu_media_list       *pWorkDirectoryList;
 struct sqlu_media_list       *pCopyTargetList;
 char                         LocalMsgFileName[] = "LOADMSG";
 char                         RemoteMsgFileName[] = "RLOADMSG";
 short                        CallerAction;
 long                         *pNullIndicators;
 void                         *pReserved;


 EXEC SQL BEGIN DECLARE SECTION;
    char userid[9];
    char passwd[19];
 EXEC SQL END DECLARE SECTION;

 printf ("This is sample program 'tload.sqc'\n");

 /* need to preset the size of structure field and counts */
 outputInfo.sizeOfStruct = SQLUEXPT_OUT_SIZE;

 /******************************************************************\
 * need to allocate the proper amount of space for the SQL statement *
 \******************************************************************/
 ActionString = (struct sqlchar *)malloc(strlen(statement)
    + sizeof (struct sqlchar));
 ActionString->length = strlen(statement);
 strncpy (ActionString->data, statement, strlen(statement));

 FileTypeMod = (struct sqlchar *)malloc(strlen("lobsinfile")
    + sizeof (struct sqlchar));
 FileTypeMod->length = strlen("lobsinfile");
 strncpy (FileTypeMod->data, "lobsinfile", FileTypeMod->length);

 /* DELimited format can not have specified names, therefore the
    column method is 'D'efault */
 DataDescriptor.dcolmeth = SQL_METH_D;

 if (argc == 1) {
    EXEC SQL CONNECT TO sample;
CHECKERR ("CONNECT TO SAMPLE");
 }
 else if (argc == 3) {
    strcpy (userid, argv[1]);
    strcpy (passwd, argv[2]);
    EXEC SQL CONNECT TO sample USER :userid USING :passwd;
    CHECKERR ("CONNECT TO SAMPLE");
 }
 else {
    printf ("\nUSAGE: tload [userid passwd]\n\n");
    return 1;
} /* endif */

printf ("Exporting EMP_RESUME table into file '%s'\n", datafile);
/*******************\
* EXPORT API called *
\*******************/
sqluexpr (datafile, NULL, NULL, &DataDescriptor, ActionString,
   FileType, FileTypeMod, msgfile_x, 0, &outputInfo, NULL, &sqlca);
CHECKERR ("exporting table");
printf ("Rows exported %d\n", outputInfo.rowsExported);


free (ActionString);

/* need to allocate the proper amount of space for the SQL statement */
ActionString = (struct sqlchar *)malloc(strlen(impStatement)
   + sizeof (struct sqlchar));
ActionString->length = strlen(impStatement);
strncpy (ActionString->data, impStatement, strlen(impStatement));

printf ("Creating a temporary table 'loadtable' to load into\n");
/* create a temporary table to import into */
EXEC SQL CREATE TABLE loadtable (num CHARACTER(6), format VARCHAR(10),
   photo BLOB(100K));
CHECKERR ("CREATE TABLE");

/* end the transaction so the program can quiesce the tablespace */
EXEC SQL COMMIT;

printf ("Quiescing tablespaces for table 'loadtable'\n");
/******************************\
* QUIESCE TABLESPACE FOR TABLE *
\******************************/
sqluvqdp ("loadtable", SQLU_QUIESCEMODE_EXCLUSIVE, NULL, &sqlca);
CHECKERR ("QUIESCE TABLESPACES FOR TABLE");


printf ("Loading the file '%s' into 'loadtable'\n", datafile);

/* initializing the variables for the LOAD API */
/* the DataFileList structure */
DataFileList.media_type = SQLU_SERVER_LOCATION;
DataFileList.sessions = 1;
DataFileList.target.location = (sqlu_location_entry *) malloc
   (sizeof(sqlu_location_entry) * DataFileList.sessions);
strcpy (DataFileList.target.location->location_entry, datafile);

pLobPathList = NULL;
CallerAction = SQLU_INITIAL;

/* the sqluload input structure */
InputInfo.sizeOfStruct = SQLULOAD_IN_SIZE; /* this should never change */
InputInfo.savecnt = 1;                     /* consistency points as frequent */
                                           /*   as possible */
InputInfo.restartcnt = 0;                  /* start at row 1 */
```

```
   InputInfo.rowcnt = 0;                      /* load all rows */
   InputInfo.warningcnt = 0;                  /* don't stop for warnings */
   InputInfo.data_buffer_size = 0;            /* default data buffer size */
   InputInfo.sort_buffer_size = 0;            /* default warning buffer size */
   InputInfo.hold_quiesce = 0;                /* don't hold the quiesce */
   InputInfo.restartphase = ' ';              /* ignored anyway, but must */
                                              /*   be ' ',L,B,D */
   InputInfo.statsopt = SQLU_STATS_NONE;      /* don't bother collecting them */

   /* the sqluload output structure */
   OutputInfo.sizeOfStruct = SQLULOAD_OUT_SIZE;

   /* the CopyTargetList structure */

      pCopyTargetList = NULL;

   OutputInfo.sizeOfStruct = SQLULOAD_OUT_SIZE;
   /******\
   * LOAD *
   \******/
   sqluload (&DataFileList,
             pLobPathList,
             &DataDescriptor,
             ActionString,
             FileType,
             FileTypeMod,
             LocalMsgFileName,
             RemoteMsgFileName,
             CallerAction,
             &InputInfo,
             &OutputInfo,
             pWorkDirectoryList,
             pCopyTargetList,
             pNullIndicators,
             pReserved,
             &sqlca);
   CHECKERR ("LOADing table");

   printf ("Rows loaded %d\nrows committed %d\n", OutputInfo.rowsLoaded,
      OutputInfo.rowsCommitted);

   free (ActionString);

   /* drop the table  */
   EXEC SQL DROP TABLE loadtable;

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
}
/* end of program : tload.sqc */
```

The source file for a sample program called loadqry.sqc can be found in the \sqllib\samples\c directory. This sample program shows how to use an API to query the current status of a load operation against a database to which the program is connected. It contains both DB2 APIs and embedded SQL calls. The script file bldvaemb.cmd, located in the same directory, contains the commands to build this and other sample programs. For general information about creating applications containing DB2 administrative APIs, and detailed information about compile and link options, see the *Application Building Guide*. To build the sample program loadqry from the source file loadqry.sqc on OS/2:

1. Copy the files loadqry.sqc, bldvaemb.cmd, util.c, and util.h to a working directory.
2. If the database manager is not running, issue the command db2start.
3. Enter bldvaemb loadqry sample. The following files are generated:

```
loadqry.bnd
loadqry.c
util.obj
loadqry.obj
loadqry.exe
```

To run the sample program (executable file), enter loadqry. You might find it useful to examine the message file. This file will contain information only if the program is run when there is a load operation in progress.

Pending States After a Load Operation

Since regular logging is not performed, the load utility uses *pending* states to preserve database consistency. These states can be checked by using the LIST TABLESPACES command (see the *Command Reference*).

The load and build phases of the load process place any associated table spaces into load pending state. To remove the load pending state (if the load operation has failed, or was interrupted):

- Terminate the load operation.
- Restart the load operation.
- Invoke a LOAD REPLACE operation against the same table on which a load operation has failed.
- Recover table spaces for the loading table by using the RESTORE DATABASE command with the most recent table space or database backup, and then carry out further recovery actions.
- Drop and then recreate table spaces for the loading table.

The delete phase places any associated table spaces into delete pending state.

Table spaces are placed in backup pending state if the load process completes, and:

- The database configuration parameter *logretain* is set to recovery, or *userexit* is enabled, and
- The load option COPY YES is not specified, and
- The load option NONRECOVERABLE is not specified.

The fourth possible state associated with the load process pertains to referential and check constraints. If an existing table is a parent table containing a primary key referenced by a foreign key in a dependent table, replacing data in the parent table places the dependent *table* (not the table space) in check pending state. To validate a table for referential integrity and check constraints, issue the SET INTEGRITY statement after the load process completes, if the table has been left in check pending state. For more information about the check pending state, see “Checking for Constraints Violations” on page 65.

Optimizing Load Performance

The performance of the load utility depends on the nature and the quantity of the data, the number of indexes, and the load options specified.

Unique indexes reduce load performance if duplicates are encountered. In most cases, it is still more efficient to create indexes during the load operation than to invoke the CREATE INDEX statement for each index after the load operation completes (see Figure 5).

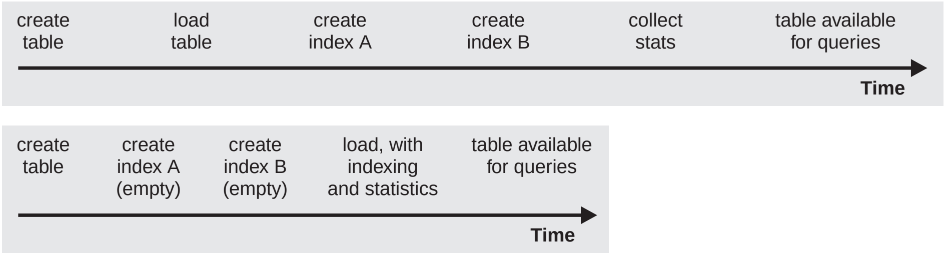


Figure 5. Increasing Load Performance through Concurrent Indexing and Statistics Collection.. Tables are normally built in three steps: data loading, index building, and statistics collection. This causes multiple data I/O during the load operation, during index creation (there can be several indexes for each table), and during statistics collection (which causes I/O on the table data and on all of the indexes). A much faster alternative is to let the load utility complete all of these tasks in one pass through the data.

When tuning index creation performance, the amount of memory dedicated to the sorting of index keys during a load operation is controlled by the *sortheap* database configuration parameter. For example, to direct the load utility to use 4000 pages of main memory for index key sorting, set the *sortheap* database configuration parameter to be 4000 pages, disconnect all applications from the database, and then issue the LOAD command.

Load performance can be improved by installing high performance sorting libraries from third party vendors to create indexes during the load operation. An example of a third party sort product is SyncSort. Use the **DB2SORT** environment variable (registry value) to specify the location of the sorting library that is to be loaded at run time. For more information about environment variables, see the *Administration Guide*.

Use of the SET INTEGRITY statement may lengthen the total time needed to load a table and make it usable again. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement will check the table for constraints violations incrementally (by checking only the appended portion of the table). If a table cannot be checked for constraints violations incrementally, the entire table is checked, and it may be some time before the table is usable again.

The load utility performs equally well in INSERT mode and in REPLACE mode.

The utility attempts to deliver the best performance possible by determining optimal values for DISK_PARALLELISM, CPU_PARALLELISM, and DATA_BUFFER, if these parameters have not be specified by the user. Optimization is done based on the size and the free space available in the utility heap. Consider allowing the load utility to choose values for these parameters before attempting to tune them for your particular needs.

Following is information about the performance implications of various options available through the load utility:

ANYORDER

Specify this file type modifier to suspend the preservation of order in the data being loaded, and improve performance. If the data to be loaded is presorted, `anyorder` may corrupt the presorted order, and the benefits of presorting will be lost for subsequent queries.

BINARY NUMERICS and PACKED DECIMAL

Use these file type modifiers to improve performance when loading positional numeric ASC data into fixed-length records.

COPY YES or NO

Use this parameter to specify whether a copy of the input data is to be made during a load operation. COPY YES reduces load performance, because all of the loading data is copied during the load operation (forward recovery must be enabled); the increased I/O activity may increase the load time on an I/O-bound system.

Specifying multiple devices or directories (on different disks) can offset some of the performance penalty resulting from this operation. COPY NO may reduce overall performance, because if forward recovery is enabled, the table is placed in backup pending state, and the database, or selected table spaces, must be backed up before the table can be accessed.

CPU_PARALLELISM

Use this parameter to exploit intra-partition parallelism (if this is part of your machine's capability), and significantly improve load performance. The parameter specifies the number of processes or threads used by the load utility to parse, convert, and format data records. The maximum number allowed is 30. This parameter is particularly useful when loading presorted data, because record order in the source data is preserved (see Figure 6). If there is insufficient memory to support the specified value, the utility adjusts the value. If this parameter is not specified, the load utility selects a default value that is based on the number of CPUs on the system.

If tables include either LOB or LONG VARCHAR data, CPU_PARALLELISM is set to one. Parallelism is not supported in this case.

Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, you may not obtain any discernible performance benefit from using it in non-SMP environments.

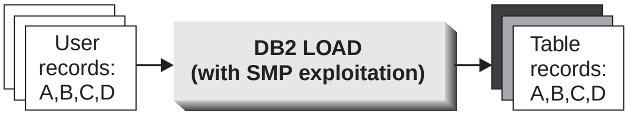


Figure 6. Record Order in the Source Data is Preserved When Intra-partition Parallelism is Exploited During a Load Operation

DATA BUFFER

The DATA BUFFER parameter specifies the total amount of memory allocated to the load utility as a buffer. It is recommended that this buffer be several *extents* in size. An extent is the unit of movement for data within DB2, and the extent size can be one or more 4KB pages. The DATA BUFFER parameter is useful when working with large objects (LOBs); it reduces I/O waiting time. The data buffer is allocated from the utility heap. Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2 utilities. The database configuration parameter *util_heap_sz* can be modified accordingly. For information about the UPDATE DATABASE CONFIGURATION command, see the *Command Reference*. The default value for the Utility Heap Size configuration parameter is 5 000 4KB pages. Because load is only one of several utilities that use memory from the utility heap, it is recommended that no more than fifty percent of the pages defined by this parameter be available for the load utility, and that the utility heap be defined large enough. For more information about *util_heap_sz*, see the *Administration Guide*.

DISK_PARALLELISM

The DISK_PARALLELISM parameter specifies the number of processes or threads used by the load utility to write data records to disk. Use this parameter to exploit available containers when loading data, and significantly improve load performance. The maximum number allowed is the greater of four times the CPU_PARALLELISM value (actually used by the load utility), or 50. By default, DISK_PARALLELISM is equal to the sum of the table space containers on all table spaces containing objects for the table being loaded, except where this value exceeds the maximum number allowed.

FASTPARSE

Use the `fastparse` file type modifier to reduce the data checking that is performed on user-supplied column values, and enhance performance. This option should only be used when the data being loaded is known to be valid. It can improve performance by about 10 or 20 percent.

NONRECOVERABLE

Use this parameter if you do not need to be able to recover load transactions against a table. Load performance is enhanced, because no additional activity beyond the movement of data into the table is required, and the load operation completes without leaving the table spaces in backup pending state.

Note: When these load transactions are encountered during subsequent restore and roll-forward recovery, the table is not updated, and is marked ″invalid″. Further actions against this table are ignored. After the roll-forward operation is complete, the table can only be dropped.

NOROWWARNINGS

Use the `norowwarnings` file type modifier to suppress the recording of warnings about rejected rows, and enhance performance, if you anticipate a large number of warnings.

SAVECOUNT

Use this parameter to set an interval for the establishment of consistency points during a load operation. The synchronization of activities performed to establish a consistency point takes time. If done too frequently, there will be a noticeable reduction in load performance. If a very large number of rows is to be loaded, it is recommended that a large SAVECOUNT value be specified (for example, a value of ten million in the case of a load operation involving 100 million records).

A LOAD RESTART operation will automatically continue from the last consistency point.

STATISTICS YES

Use this parameter to collect data distribution and index statistics more efficiently than through invocation of the runstats utility following completion of the load operation, even though performance of the load operation itself will decrease (particularly when DETAILED INDEXES ALL is specified).

For optimal performance, applications require the best data distribution and index statistics possible. Once the statistics are updated, applications can use new access paths to the table data based on the latest statistics. New access paths to a table can be created by rebinding the application packages using the DB2 BIND command (see the *Command Reference*).

When loading data into large tables, it is recommended that a larger value for the *stat_heap_sz* (Statistics Heap Size) database configuration parameter be specified. For information about the UPDATE DATABASE CONFIGURATION command, see the *Command Reference*. For more information about *stat_heap_sz*, see the *Administration Guide*.

WARNINGCOUNT

Use this parameter to specify the number of warnings that can be returned by the utility before a load operation is forced to terminate. If you are expecting only a few warnings or no warnings, set the WARNINGCOUNT parameter to approximately the number you are expecting, or to twenty if you are expecting no warnings. The load operation will stop after the WARNINGCOUNT number is reached. This gives you the opportunity to correct data (or to drop and then recreate the table being loaded) before attempting to complete the load operation. Although not having a direct effect on the performance of the load operation, the establishment of a WARNINGCOUNT threshold prevents you from having to wait until the entire load operation completes before determining that there is a problem.

Restrictions and Limitations

The following restrictions apply to the load utility:

- This utility does not support the use of nicknames.
- Loading data into typed tables is not supported.
- Attempts to create or to drop tables in a table space that is in load pending state will fail.
- You cannot load data into a database accessed through DB2 Connect or a down-level server prior to DB2 Version 2. Options that are only available with this release of DB2 cannot be used with a server from the previous release.
- If an error occurs during a LOAD REPLACE operation, the original data in the table is lost. Retain a copy of the input data to allow the load operation to be restarted.
- Triggers are not activated on newly loaded rows. Business rules associated with triggers are not enforced by the load utility.
- When running concurrent load operations, the temporary files path names must be unique. If they are not unique (for example, if the TEMPFILES PATH parameter was not specified on any of the concurrently running load jobs), the load utility does not enforce uniqueness, and inconsistent results will be obtained.

Troubleshooting

During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter.

These message files are standard ASCII text files. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Notes:

1. You can only view the contents of a message file after the operation is finished.
2. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility.

Chapter 4. AutoLoader

This chapter describes the DB2 UDB AutoLoader utility, which can be used in a partitioned database environment to load data across all or some of the partitions at the same time.

The following topics are covered:

- "AutoLoader Overview"
- "Privileges, Authorities, and Authorization Required to Use AutoLoader" on page 132
- "Using AutoLoader" on page 133
- "Loading into Multiple Database Partitions" on page 133
- "AutoLoader Options" on page 134
- "Example AutoLoader Session" on page 141
- "Migration and Back-level Compatibility" on page 143
- "AutoLoader Hints and Tips" on page 144
- "Restrictions and Limitations" on page 145
- "AutoLoader Troubleshooting" on page 146.

AutoLoader Overview

The AutoLoader is a utility that can:

- Transfer data from one system (like MVS) to another system (like UNIX).
- Partition that data in parallel.
- Load the data simultaneously on the corresponding database partitions.

The AutoLoader can be run in one of four modes:

- SPLIT_AND_LOAD. Data is partitioned (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- SPLIT_ONLY. Data is partitioned (perhaps in parallel) and the output is written to files in a specified location, or in the AutoLoader current working directory.
- LOAD_ONLY. Data is assumed to be already partitioned; the split process is skipped, and the data is loaded simultaneously on the corresponding database partitions.

- ANALYZE. An optimal partitioning map with even distribution across all database partitions is generated.

In a partitioned database, large amounts of data are located across many partitions. Partitioning keys are used to determine on which database partition each portion of the data resides. The data must be *split* before it can be loaded at the correct database partition. The AutoLoader utility can perform both operations (see Figure 7).

The AutoLoader utility uses a hashing algorithm to partition the data into as many output sockets as there are database partitions in the nodegroup in which the table was defined. It then loads from these output sockets concurrently across the set of database partitions in the nodegroup. A key feature of this utility is that it uses direct TCP/IP communication using sockets for all data transfer required during both split and load processes. It also allows the use of multiple database partitions for the splitting phase, thereby significantly improving performance.

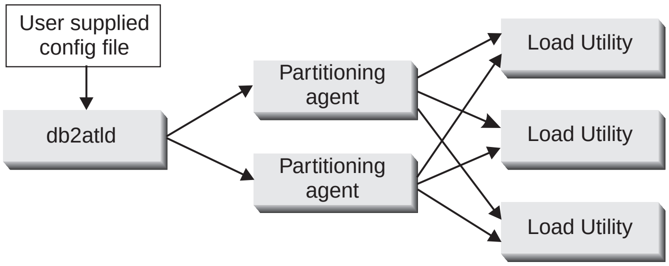


Figure 7. AutoLoader Overview.. In this example, source data is read by the AutoLoader, and half is sent to each of two partitioning agents, which partition the data and send it to one of three database partitions. The load utility at each partition loads the data.

Privileges, Authorities, and Authorization Required to Use AutoLoader

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

The authority required to use the AutoLoader utility is the same as that required by the load utility (see “Privileges, Authorities, and Authorization Required to Use Load” on page 63). Moreover, the load dumpfile, the remote message file, and the directory used for sorting must all be write-accessible to the instance owner.

Using AutoLoader

Before Using AutoLoader

Before invoking the AutoLoader utility:

1. Create a temporary working directory. This directory must be accessible to all participating database partitions. It is from this directory that you will invoke the AutoLoader utility.
2. Modify your AutoLoader configuration file (see “Example AutoLoader Session” on page 141), and copy it into the working directory.
3. Ensure that the *svcename* database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This is important, because the AutoLoader utility makes remote database connections from the working partition (from which you invoke the utility) to the database partitions (on which the table is defined).

Invoking AutoLoader

The AutoLoader utility is invoked through the **db2atld** command:

```
db2atld [-config config_file] [-restart] [-terminate]

   where "-config config_file" specifies an AutoLoader configuration file
   (the default is "autoloader.cfg"); "-restart" requests restart
   of an interrupted AutoLoader operation (the configuration file
   does not need to be modified to restart); and "-terminate" requests
   termination of an interrupted AutoLoader operation.
```

A sample configuration file, autoloader.cfg, can be found in the sqllib/samples/autoloader directory. It is recommended that you copy, rename, and customize the sample configuration file according to the operations that you want the utility to perform.

Loading into Multiple Database Partitions

If you are loading data into a table in a multiple database partition nodegroup, the load utility requires that the files to be loaded are split and contain the correct header information. The load utility verifies the header information that the AutoLoader split operation writes to each data file to ensure that the data goes to the correct location.

If you are loading data into a table in a single database partition nodegroup, the files do not have to be split, even if the table is defined to have a partitioning key. In this situation, you would specify the noheader modifier for the load operation.

Loading into Multiple Database Partitions

The load utility checks that the partitioning map used by the AutoLoader split operation is the same as that specified when the table is being loaded. If not, an error is returned. It also checks that the file partition is loaded at the correct database partition, and that the data types of the partitioning key columns specified during splitting match the current definition in the catalog. The nodegroup to which the table is loaded cannot be redistributed between the time that the data file is partitioned and the time that the parts are loaded into the corresponding database table. If redistribution has been done, the utility cannot load the partitioned data.

Although the load utility supports the following flat file formats:

- Non-delimited ASCII (ASC)
- Delimited ASCII (DEL)
- PC/IXF

AutoLoader can only be used to partition ASC and DEL files. PC/IXF files cannot be split, but can be loaded into a single database partition nodegroup using the `noheader` modifier for the load operation.

The LOAD ROWCOUNT clause is not supported in an AutoLoader operation. This parameter is only valid in a non-partitioned database environment. The LOAD SAVECOUNT clause is not supported if multiple splitters are used in an AutoLoader operation.

If a column that is part of the partitioning key is invalid or rejected, none of the data associated with that row is loaded. The row is not placed in the *dumpfile*, even if one has been specified; instead, a message indicating that the record has been rejected is written to the splitter log file. Be sure to check the splitter log file after the completion of the AutoLoader process.

AutoLoader Options

There are many options that you can specify in the AutoLoader configuration file.

RELEASE Level
The release level of this configuration file. Do not delete or modify this line in the configuration file.

LOAD Command
The most important part of the configuration file is the LOAD command. The AutoLoader needs the LOAD command to direct the handling of the data, even if the selected mode of operation does not suggest that any loading is required. For example, the AutoLoader extracts useful information from the LOAD command even when

performing a SPLIT_ONLY mode operation. Specifications on the LOAD command indicate where the data is coming from, what type of data it is (delimited ASCII, for example), how the data is to be loaded, and the target table name.

Be sure to specify a complete LOAD command that includes the schema name, file name, file type, and table name. The AutoLoader utility also requires that the LOAD command conform to the format of the "db2 -f" file, except for the extra leading "db2" keyword. For detailed information about this and other command line processor (CLP) options, see the *Command Reference*. There is no need to use the special escape shell characters in the LOAD command. Finally, if the last character on a line is a backslash (\) character, the next line is a continuation of the current line. In this case, the backslash and the end-of-line characters are ignored.

For detailed information about all of the parameters available on the LOAD command, see "LOAD Command" on page 70.

DATABASE Parameter

This parameter is used to identify the database into which the data is to be loaded. If no name is specified, `SAMPLE` is used as the default value.

HOSTNAME Parameter

This parameter specifies the name of the remote machine on which the data file resides. This machine can be an MVS host or another workstation. If not specified, and the FILE_TRANSFER_CMD parameter is set, the host name `nohost` is passed to the FILE_TRANSFER_CMD parameter in the <hostname> argument. There is no default value associated with this parameter.

FILE_TRANSFER_CMD Parameter

The previous version of AutoLoader supported the concept of host file transfer, whereby the AutoLoader utility could be configured to transfer data files from a remote host. That option has been replaced by the FILE_TRANSFER_CMD option. This parameter specifies the fully qualified name of an executable file, batch file, or script that is used to transfer data from a remote host. The path must be accessible to the AutoLoader. The full path, including the execution file name, must not exceed 254 characters.

Before invoking the specified file, the AutoLoader establishes named pipes in anticipation of the data being sent from the host. The number of named pipes to be created is equivalent to the number of files or devices listed in the FROM clause on the LOAD command. This information from the LOAD command is also used to specify the parameters that are to be passed to the executable file, batch file, or script.

Based on this information, AutoLoader creates the following command:

```
<COMMAND> <logpath> <hostname> <basepipename>
   <nummedia> <source media list>
```

where

- <COMMAND> is the fully qualified path to an executable file, batch file, or script used to move data from the host.

and the remaining items are parameters that can be used by the command:

- <logpath> is the AutoLoader log path. The COMMAND program can use this path to write diagnostic or temporary data.
- <hostname> is the host name specified by the HOSTNAME parameter.
- <basepipename> is the base name for named pipes that the AutoLoader will create. The AutoLoader utility generates the base name and guarantees it to be unique on the system. The base name is appended to by the utility to create the necessary named pipes.
- <nummedia> is the number of files or devices providing data (listed in the FROM clause on the LOAD command).
- <source media list> includes the names of each of the files or devices providing data (listed in the FROM clause on the LOAD command). The names are delimited by double quotation marks to avoid potential problems caused by special characters that may be present in the names.

An AIX sample file called `atldftp.drv` can be found in the `sqllib/samples/autoloader` directory. The sample shows how FTP can be used to move data from a remote host.

SPLIT_FILE_LOCATION Parameter

This parameter is used in two ways:

- To provide the path name to the location of the split files if the utility is in LOAD_ONLY mode.
- To provide the path name to the location in which the files that have been partitioned (if the utility is in SPLIT_ONLY mode) are to be placed.

If a value for this parameter is not specified, and the utility is operating in SPLIT_ONLY mode, the split files are placed in the current working directory; if the utility is operating in LOAD_ONLY mode, it looks for the split files in the current working directory.

OUTPUT_NODES Parameter

The database partitions on which the load operation is to be performed are identified by this parameter. The specified partition numbers must be a subset of the database partitions on which the table is defined. The default value is all; that is, all database partitions on which the table is defined will have data loaded into them.

SPLIT_NODES Parameter

The database partitions participating in the splitting process are specified through this parameter. These database partitions may be the same or different from the database partitions being loaded. If a value for this parameter is not specified, the AutoLoader determines how many partitions are needed for splitting, and which partitions will be used to achieve optimal performance. The following rules are used to determine how many partitions are needed for splitting:

- If the ANYORDER modifier in the LOAD command is not specified, only one splitter is used in the AutoLoader session, and
  - If only one partition is specified through the OUTPUT_NODES parameter, or the working partition of the AutoLoader is not an element of the value specified for the OUTPUT_NODES parameter, the working partition of AutoLoader is used as the splitting partition.
  - Otherwise, the first partition other than the AutoLoader working partition (found in OUTPUT_NODES) is used as the splitting partition.
- If the `anyorder` modifier in the LOAD command *is* specified,
  1. The number of splitting partitions is determined by
     ```
     (number of partitions in OUTPUT_NODES)/4 + 1
     ```
  2. This number of partitions is chosen from those specified for the OUTPUT_NODES parameter, excluding the AutoLoader working partition.

RUN_STAT_NODE Parameter

In conjunction with the STATISTICS YES specification on the LOAD command, you can specify the database partition on which you want to collect statistics. If left blank or a value of `-1` is specified, the default value is the first database partition in the output partition list.

MODE Parameter

This parameter specifies the mode in which the AutoLoader utility is to run. Valid values are: SPLIT_AND_LOAD (the default), SPLIT_ONLY, LOAD_ONLY, or ANALYZE.

SPLIT_AND_LOAD

In this mode, data is partitioned and then loaded on the correct database partitions. Data is transferred through direct TCP/IP communication using sockets. Multiple input files are allowed.

SPLIT_ONLY

In this mode, the data is only split. A set of split data files is generated for the specified database partitions. You must have sufficient storage for each of the split data files. The split function writes the files in the location specified by the *SPLIT_FILE_LOCATION* parameter, or in the current working directory. The directory location must be write-accessible. Data is partitioned into separate files that are named using the convention *filename.xxx*, where *xxx* represents the number of the partition to which the split file belongs. If there are multiple input data files in the LOAD command, they will all be split. However, only one split file is generated for each database partition. The name of the split file is the same as the name of the first input data file.

LOAD_ONLY

In this mode, previously split data is loaded. The data is contained in separate files that are named using the convention *filename.xxx*, or *filename.*`00`*xxx*, where *xxx* represents the number of the partition to which the split file belongs. AutoLoader expects to find these files in the *SPLIT_FILE_LOCATION* or in the current working directory. The directory location must be read-accessible. The split files are loaded concurrently on their corresponding partitions. If there are multiple input data files in the LOAD command (such as `infile1`, `infile2`, and so on), AutoLoader loads `infile1.xxx` if it exists. Otherwise, it loads `infile1.00xxx` if it exists. If neither exists, AutoLoader returns an error. If both exist, AutoLoader loads `infile1.xxx`. Once the first `infile1` of either file type (*xxx* or `00`*xxx*) is loaded, checking begins for `infile2`, and this process is repeated until all of the input files are loaded.

ANALYZE

In this mode, a customized optimal partitioning map for a nodegroup is generated. It is recommended that a data file with a large number of records be specified as input (multiple input files are allowed); if this is done, the map will produce a more even distribution of data across each of the database partitions in the nodegroup. The output is written to the file specified by the *MAP_FILE_OUTPUT* parameter. The REDISTRIBUTE NODEGROUP command (see the *Command Reference*) must be invoked before the new partitioning map can take effect. Subsequent AutoLoader invocations in SPLIT_AND_LOAD mode will automatically use the new partitioning map. The *MAP_FILE_INPUT* parameter can be used when partitioning the data according to the new partitioning map without changing the default partitioning map of the nodegroup.

LOGFILE Parameter

This parameter is used to provide the base name of the temporary and permanent files used by the AutoLoader utility:

```
<logfile>.split.cfg ...
   Configuration file for all splitters.
<logfile>.split.<3-digit-node-number>.log ...
   Log file for each splitter.
<logfile>.pmap.<pid> ...
   Internal temporary file, where <pid> is the process ID
      of this AutoLoader job.
<logfile>.load.<3-digit-node-number> ...
   Message file for each loading process if there is no
      message file specified in the LOAD command.
```

Although you can specify a path for the LOGFILE parameter, you must verify the existence and accessibility of that path. The default value is `./autoloader.log`.

Note: If there are multiple concurrent AutoLoader sessions, you must ensure that the specified base name or the path name is unique.

AUTHENTICATION and PASSWORD Parameters

These parameters are necessary if a password is required for remote invocation of the splitter program, or client/server database connections when loading. The default value for AUTHENTICATION is `NO` (no password checking), and any value specified for the PASSWORD parameter is ignored.

The concept of a local database connection has been extended for MPP environments to include connections from any node of a given MPP instance. That means that even though the instance is configured using AUTHENTICATION server, a password is not required if a connection is being attempted from one of the nodes defined in the `db2nodes.cfg` file. The AutoLoader makes use of this new connection behavior when the AUTHENTICATION flag in the AutoLoader configuration file is not set, or is set to `NO`, and a value for the PASSWORD parameter is not specified. A password for AutoLoader is only mandatory if a password is required for remote execution of programs on your system. For example, a password is required if the `.rhosts` file on a UNIX system has not been set up properly to enable **rsh** execution.

Alternatively, if a password is needed, the DB2 registry variable **DB2ATLD_PWFILE**, which defines the fully qualified path to a password file created by the user, can be set. Both the password file and the fully qualified path must be accessible to the AutoLoader utility. If this variable is defined, the first word in the file pointed to by its value will be the password.

MAX_NUM_SPLITTERS Parameter

This parameter specifies the maximum number of splitter processes that can be used in an AutoLoader job. The default value is 25.

FORCE Parameter

This parameter forces the AutoLoader job to continue even if the utility determines (at startup time) that some target partitions or table spaces are offline. If the value is `NO`, and some partitions are unavailable, no data will be processed. If the value is `YES`, database partitions that are available will be loaded, and all others will be ignored. The default value for this parameter is `NO`.

STATUS_INTERVAL Parameter

This parameter specifies the number of megabytes (MB) of data to load before generating a progress message. Valid values are whole numbers in the range of 1 to 4000. The default value is 100.

PORTS Parameter

This parameter specifies the range of TCP ports used to create sockets for internal AutoLoader communications. The default range is 6000 to 6063. If defined at the time of AutoLoader invocation, the value of the **DB2ATLD_PORTS** DB2 registry variable replaces any value specified for this parameter.

CHECK_LEVEL Parameter

This parameter specifies whether checking for record truncation during input or output should be performed. Valid values are `CHECK` and `NOCHECK`. The default value is `NOCHECK`.

MAP_FILE_INPUT Parameter

This parameter specifies the name of the input file that points to a file containing the customized partitioning map. If the partitioning map is a customized (not a default) map, this parameter must be specified. You can get a customized partitioning map by invoking the AutoLoader in ANALYZE mode to generate an optimal map. This map must be moved to each database partition in your database before actual loading can proceed.

MAP_FILE_OUTPUT Parameter

This parameter specifies a name for the partitioning map when the AutoLoader is invoked in ANALYZE mode. An optimal partitioning map distributes data evenly across all database partitions. If a value for this parameter is not specified, and the utility is running in ANALYZE mode, an error is returned.

TRACE Parameter

This parameter specifies the number of records to trace when you need to review a dump of all of the data conversion process and the output of hashing values. The default value is zero (no tracing).

NEWLINE Parameter

This parameter specifies the character that is used to delimit each record in the data file. This parameter is meaningful only if the input data file is a fixed-length ASC file with each record delimited by a new line character, and the `reclen` modifier in the LOAD command has been specified. If a value of `YES` is specified, the AutoLoader always checks whether the record is terminated by a new line character. It also checks whether the record length matches that specified through the `reclen` modifier. The default value is `NO`.

Example AutoLoader Session

Following is a sample AutoLoader configuration file (on AIX):

```
###############
# release level
###############
RELEASE=V6.00

##################
# CLP load command
##################
db2 load from /home/user/atld_work/test.dat of del replace into user.test

###############
# database name
###############
database=wsdb

#################
# split partition list
#################
SPLIT_NODES=(0,2)

##############
# running mode
##############
mode=split_and_load

################
# log file token
################
logfile=mylog
```

```
######################################
# frequency of progressive information
#
# print out progressive info every 10
# mega-bytes of data
######################################
STATUS_INTERVAL=10
```

The following command issued against this configuration file includes the path and the temporary working directory accessible from each of the participating database partitions. The name of the configuration file, `sample.atld.cfg`, is also specified:

```
/home/user/atld_work/ $ db2atld -config sample.atld.cfg
```

Invocation of this command produces the following output:

```
/home/user/atld_work/ $ db2atld -config sample.atld.cfg
Utility program: "db2atld". Version: "06000".
Start reading autoloader configuration file: sample.atld.cfg
Finish reading autoloader configuration file: sample.atld.cfg
Start initializing autoloader process.
Finish initializing autoloader process.
The Autoloader is now issuing all LOAD requests.
The LOAD operation has begun on partition "0".
The LOAD operation has begun on partition "1".
The LOAD operation has begun on partition "2".
The LOAD operation has begun on partition "3".
The Autoloader is now issuing all split requests.
Start db2split on node "0" in background.
Start db2split on node "2" in background.
The utility has read "10" megabytes from the source data.
The utility has read "20" megabytes from the source data.
The utility has read "30" megabytes from the source data.
The utility has read "40" megabytes from the source data.
The utility has read "50" megabytes from the source data.
The utility has read "60" megabytes from the source data.
The utility has read "70" megabytes from the source data.
The utility has read "80" megabytes from the source data.
The utility has read "90" megabytes from the source data.
The utility has read "100" megabytes from the source data.
The utility has read "110" megabytes from the source data.
The utility has read "120" megabytes from the source data.
The utility has read "130" megabytes from the source data.
The utility has completed reading "130" megabytes from the user data.
The Autoloader is waiting for all splitters to complete.
The Autoloader is waiting for all LOAD operations to complete.
The remote execution of the splitter utility on partition "2"
finished with remote execution code "0".
The remote execution of the splitter utility on partition "0"
finished with remote execution code "0".
```

```
  Operation      Node     SQL Code     Result
____________________________________________________________
  LOAD           000      +00000000    Success.
____________________________________________________________
  LOAD           001      +00000000    Success.
____________________________________________________________
  LOAD           002      +00000000    Success.
____________________________________________________________
  LOAD           003      +00000000    Success.
____________________________________________________________
  SPLIT          000      +00000000    Success.
____________________________________________________________
  SPLIT          002      +00000000    Success.
____________________________________________________________
  PSPLIT         000      +00000000    Success.
____________________________________________________________
  RESULTS:       4 of 4 LOADs completed successfully.
____________________________________________________________

Rows Read      1310848
Rows Skipped   0
Rows Loaded    1310848
Rows Rejected  0
Rows Deleted   0
Rows Committed 1310848
```

The main body of the messages generated by the AutoLoader pertain to the initialization of the participating database partitions. Both splitting and loading processes are logged. Termination of the AutoLoader processes is also recorded.

A summary table of operations performed, partitions used, SQL codes returned, and results obtained, is also generated. If an SQL code other than zero is returned, a review of the message file will show the specific warnings or errors that were recorded.

A record summary for the AutoLoader job completes the output.

Migration and Back-level Compatibility

There are migration and back-level compatibility issues associated with the AutoLoader utility:

- An earlier version of the AutoLoader utility was invoked through the **db2autold** command. The current version is invoked through the **db2atld** command.
- **db2atld** uses sockets as internal communications channels (as opposed to named pipes), and it chooses a TCP port number from the default range of 6063 down to 6000. However, if your system requires this range for other applications, you have two options when you migrate:

  - The PORTS parameter in the AutoLoader configuration file can be used to specify a port range other than the default.
  - The **DB2ATLD_PORTS** DB2 registry variable can be defined by specifying the range as:

    ```
    <lower-port-number>:<higher-port-number>
    ```

  The priority sequence for determining the TCP port range is: the **DB2ATLD_PORTS** DB2 registry variable, the PORTS AutoLoader configuration parameter, and the default.

- If a password is needed for client-to-server database connections, you have two options when you migrate:
  - The AutoLoader configuration parameters AUTHENTICATION and PASSWORD can be used. If AUTHENTICATION is set to `YES`, and PASSWORD is defined, the password is used for authentication. If AUTHENTICATION is set to `YES`, and PASSWORD is not defined, you are prompted for a password.
  - The DB2 registry value **DB2ATLD_PWFILE** can be set to point to a file where the password is stored. If specified, the contents of the file are evaluated, and the first blank-delimited character string is used as the password. Since the registry value is evaluated last, if defined, it will be used to override other password values.

AutoLoader Hints and Tips

Following is some information to consider before using the AutoLoader utility:

- Familiarize yourself with AutoLoader operations by using the utility with small amounts of data.
- If the input data is already sorted, or in some chosen order, and you wish to maintain that order during the loading process, only one database partition should be used for splitting. Parallel splitting cannot guarantee that the data will be loaded in the same order it was received.
- If large objects (LOBs) are being loaded from separate files (that is, if you are using the `lobsinfile` modifier through the load utility), all directories containing the LOB files must be read-accessible to all the database partitions where loading is taking place. The LOAD *lob-path* parameter must be fully qualified when working with LOBs.
- All temporary AutoLoader files reside in the directory specified through the LOGFILE AutoLoader configuration parameter. This directory must be network-accessible with both read and write access to all partitions where splitting is to be done. By specifying different directories for temporary files, you can run multiple concurrent AutoLoader jobs to load data into separate tables in different table spaces.

- The maximum number of active database connections in an AutoLoader job is the number of loading partitions defined in the OUTPUT_NODES AutoLoader configuration parameter. Ensure that the *maxxappls* (maximum number of active applications) database configuration parameter has been set high enough.
- You can force an AutoLoader job to continue even if the AutoLoader detects (at startup time) that some loading partitions or associated table spaces are offline, by specifying `FORCE=YES` in the AutoLoader configuration file.
- Use the STATUS_INTERVAL AutoLoader configuration parameter to monitor the progress of an AutoLoader job. AutoLoader returns messages at specified intervals, indicating how many megabytes of data have been processed.
- Better performance can be expected if the splitting partitions (as defined by the SPLIT_NODES parameter) are different from the loading partitions (as defined by the OUTPUT_NODES parameter), since there is less contention for CPU cycles. The AutoLoader utility itself should be invoked on a database partition that is not participating in either the splitting or the loading operation. On an SMP system, you can improve performance by ensuring that there is at least one splitter task for every available CPU.
- AutoLoader ignores the MESSAGES parameter in the LOAD command, and directs all messages from the LOAD command into the file `load_log.`*XXX*; this file contains messages from the load process on database partition *XXX*. AutoLoader also creates a file called `splt_log.`*XXX*; this file contains messages from the split process on database partition *XXX*. The utility also creates a file called `autoload.log`, containing messages from the main AutoLoader script. Check it to ensure that all pipes and temporary directories have been set up correctly.
- AutoLoader chooses only one output database partition on which to collect statistics. The RUN_STAT_NODE AutoLoader configuration parameter can be used to specify that partition.
- Multiple invocations of AutoLoader can be used to load data simultaneously into separate tables. Ensure that:
  - The tables reside in separate table spaces.
  - All of the AutoLoader operations are invoked from separate directories.
  - The data file name used to create temporary pipes is unique for each AutoLoader operation.

Restrictions and Limitations

The following restrictions apply to the AutoLoader utility:

- The location of the input files to the load operation cannot be a tape device.

AutoLoader Restrictions and Limitations

- AutoLoader does not support the ROWCOUNT option on the LOAD command.
- If you are using multiple database partitions to partition and then load the data, the use of a SAVECOUNT greater than zero on the LOAD command is not supported.

AutoLoader Troubleshooting

If it appears that the AutoLoader utility is hanging, you can:

- Use the STATUS_INTERVAL parameter of the AutoLoader configuration file to monitor the progress of an AutoLoader job.
- Check the `<logfile>.split.<3-digit-node-number>.log` files to see the status of the splitter processes on each splitting database partition. If things are going well, and the TRACE parameter in the AutoLoader configuration file has been set, there should be trace messages for a certain number of records in these log files.
- Check the LOAD messages file or the `<logfile>.load.<3-digit-node-number>` files to see if there are any load error messages.
- Interrupt the current AutoLoader job if you find errors suggesting that one of the AutoLoader processes encountered errors.

If the AutoLoader utility is still failing, you can:

1. Set the MODE parameter in the AutoLoader configuration file to `SPLIT_ONLY`, and invoke the utility again.
2. Check the split data files to see if there is anything abnormal in them. If the split files look correct, try to manually load one of those split files on the correct database partition.
3. If the data loads correctly, there might be additional AutoLoader problems or database system problems. Contact your IBM service representative.

The following applies to an error scenario for the AutoLoader utility on IBM DB2 Universal Database Enterprise - Extended Edition for Windows NT.

When running **db2atld** on a multi-homed machine (that is, a machine with multiple network cards installed), ensure that the machine is configured correctly by typing the **hostname** command on the machine where the AutoLoader is running, and then pinging this host name from the same machine. The IP address returned should be the same as that returned when this host name is pinged from another machine in your DB2 MPP node list. If the machine is not configured correctly, the utility returns an SQL6555N error, and you will see the error message `errno = 10061` (connection refused) in the `db2diag.log` files on some of the loading nodes defined by the OUTPUTNODES parameter in your AutoLoader configuration file.

On Windows NT machines, the IP address returned for a local host name is not retrieved from the `DNS` or the `hosts` file, but from information configured locally in the Control Panel network icon. A Windows NT Version 4.0 defect causes the IP address order returned on a multi-homed machine to ignore the binding order configured in the Control Panel network icon. See Microsoft Support online article Q171320 for information that will help you to solve this problem.

Chapter 5. Moving DB2 Data Links Manager Data

This chapter describes how to use the DB2 export, import, and load utilities to move DB2 Data Links Manager data.

For information about the file formats that you can use with these utilities, see "Appendix C. Export/Import/Load Utility File Formats" on page 179.

For detailed information about the DB2 Data Links Manager, see the *DB2 Data Links Manager for Windows NT Quick Beginnings*, or the *DB2 Data Links Manager for AIX Quick Beginnings*.

The following topics are covered:

- "Using Export to Move DB2 Data Links Manager Data"
- "Using Import to Move DB2 Data Links Manager Data" on page 152
- "Using Load to Move DB2 Data Links Manager Data" on page 153.

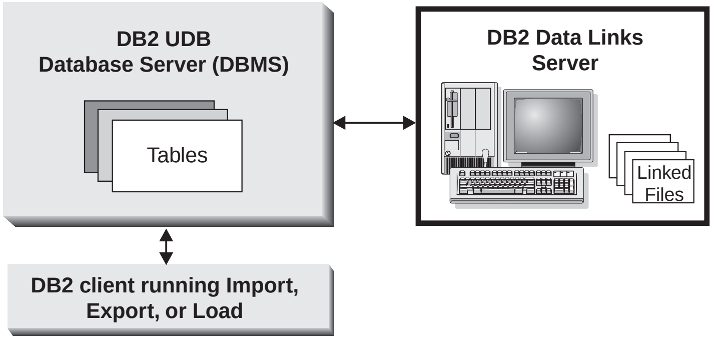


Figure 8. Moving DB2 Data Links Manager Data.. Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the export, import, and load utilities must move both the database data, and the data files on the corresponding Data Links servers.

Using Export to Move DB2 Data Links Manager Data

Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the export utility must move both the database data, and the data files on the corresponding Data Links servers (see Figure 8). To do this, the export utility produces one control file per Data Links server. The name of the control file is the same as the name of the Data Links server. The control files are created in a new directory that has the name dlfm/*YYYYMMDD/HHMMSS*, where *YYYYMMDD* represents *YearMonthDay*, and *HHMMSS* represents *HourMinuteSecond*. This directory is created under the same directory in which the export data file is created. A control file lists the names of the corresponding DB2 Data Links Manager files that are referenced by the DATALINK columns of the rows that are exported.

On the WINDOWS NT operating system, the export utility produces only one control file for all Data Links servers. The name of this control file is `ctrlfile.lst`. It is created in a new directory that has the name dlfm\YYYYMMDD\HHMMSS. This directory is created under the same directory in which the export data file is created. The control file lists the URLs of all DB2 Data Links Manager files that are referenced by the DATALINK columns of the rows that are exported.

DATALINK values that have the NO LINK CONTROL property are not placed in the control file.

The control files must be transported to their respective Data Links servers. On the Windows NT operating system, the single control file must be transported to all referenced Data Links servers. The **dlfm_export** utility should be run at each Data Links server, specifying the control file name. This utility produces an archive of the files listed in the control file for that Data Links server.

To ensure that a consistent copy of the table and the corresponding files that are referenced by the DATALINK columns are copied, perform the following steps:

1. Ensure that no update transactions are in progress when the export operation is running by issuing the following command:
   ```
   db2 quiesce tablespaces for table tablename share
   ```
2. Invoke the export utility.
3. Run the **dlfm_export** utility with root authority at each Data Links server; this will successfully archive files to which the Data Links File Manager administrator may not have access. As input to **dlfm_export**, specify the name of the control file that was generated by the export utility.
4. Make the table available for updates by issuing the following command:
   ```
   db2 quiesce tablespaces for table tablename reset
   ```

The export utility executes as an SQL application. The rows and columns that satisfy the conditions of the SELECT statement are extracted from the database. For DATALINK columns, the SELECT statement should not specify any scalar functions.

The export utility generates the following files:

- The export data file. A DATALINK column value in this file has the same format as that used by the import and the load utilities. If the DATALINK column value is NULL, it is treated in the same way as are other NULL columns.
- Control files for each Data Links server. The control file lists the complete path and the names of all the files that are to be exported from that Data Links server. On the Windows NT operating system, there is only one control file for all Data Links servers referenced by DATALINK column values.

Use the **dlfm_export** utility to export files from a Data Links server as follows:

```
   dlfm_export control-file-name archive-file-name
```

where *control-file-name* is the name of the control file generated by running the export utility on the DB2 client, and *archive-file-name* is the name of the archive file that will be generated. The default *archive-file-name* is `export.tar`, located in the current working directory.

A complementary utility called **dlfm_import** is provided to retrieve and restore files from the archive that **dlfm_export** generates. This utility must be used whether the archived files are being restored on the same, or a different, Data Links server.

Use the **dlfm_import** utility to retrieve files from the archive as follows:

```
   dlfm_import archive-file-name
```

where *archive-file-name* is the name of the archive file that will be used to restore the files. The default *archive-file-name* is `export.tar`. Run the **dlfm_import** utility with root authority at each Data Links server, because you may want to restore the archived files on a different Data Links server, which may not have the same directory structure and user IDs as the Data Links server on which the **dlfm_export** utility was run.

Notes:

1. The DB2 Data Links Manager does not have to be running when you invoke these utilities.
2. When running the **dlfm_import** utility on a Data Links server other than the one on which the **dlfm_export** utility was run, the files will be restored in the correct paths. The files will be owned by root in case some of the user IDs do not exist on the importing machine. Before inserting these files into a database, ensure that all files have the correct permissions and belong to the correct user IDs.

Using Export to Move DB2 Data Links Manager Data

The following table shows how to export the DB2 data and the files that are referenced by the database called `SystemA` to the database called `SystemB`. SystemA uses the Data Links servers DLFM1 and DLFM2. SystemB uses the Data Links servers DLFMX and DLFMY. The files on DLFM1 will be exported to DLFMX, and the files on DLFM2 will be exported to DLFMY.

Database SystemA with Data Links Servers DLFM1 and DLFM2			**Step**
DB2 data on File	File1 with file names for DLFM1	File2 with file names for DLFM2	1) Run the dlfm_export command (as root) on both Data Links servers. This will produce an archive on both Data Links servers.
Database SystemB with Data Links Servers DLFMX and DLFMY			
	On DLFMX, restore from archive	On DLFMY, restore from archive	2) Run dlfm_import (as root) on both Data Links servers.
			3) Run the IMPORT command on SystemB, using the parameter DL_URL_REPLACE_PREFIX to specify the appropriate Data Links server for each exported file.
When you run the IMPORT command on SystemB, the SystemA data and all files referenced by DATALINK columns are imported.			

Using Import to Move DB2 Data Links Manager Data

Since table data resides in the database, and the files referred to by DATALINK columns reside on Data Links servers, the import utility must move both the database data, and the data files on the corresponding Data Links servers (see Figure 8 on page 149).

Before running the import utility against the target database:

1. Copy the files that will be referenced to the appropriate Data Links servers. The **dlfm_import** utility can be used to extract files from an archive that is generated by the **dlfm_export** utility.
2. Define the prefix name (or names) to the Data Links File Managers on the Data Links servers. (You may want to perform other administrative tasks, such as registering the database.)
3. Update the Data Links server information in the URLs (of the DATALINK columns) from the exported data for the SQL table, if required. (If the Data

Links servers of the original configuration are the same as those at the target location, the Data Links server names need not be updated).

4. Define the Data Links servers at the target configuration in the DB2 Data Links Manager configuration file.

When the import utility runs against the target database, files referred to by DATALINK column data are linked on the appropriate Data Links servers.

Using Load to Move DB2 Data Links Manager Data

If you are loading data into a table with a DATALINK column that is defined with FILE LINK CONTROL, perform the following steps before invoking the load utility. (If all the DATALINK columns are defined with NO LINK CONTROL, these steps are not necessary.)

1. Ensure that DB2 Data Links Manager is installed on the Data Links servers that will be referred to by the DATALINK column values.
2. Ensure that the database is registered with the DB2 Data Links Manager.
3. Copy to the appropriate Data Links servers all files that will be inserted as DATALINK values.
4. Define the prefix name (or names) to the DB2 Data Links Managers on the Data Links servers.
5. Register the Data Links servers referred to by DATALINK data (to be loaded) in the DB2 Data Links Manager configuration file.

The connection between DB2 and the Data Links server may fail while running the load utility, causing the load operation to fail. If this occurs:

1. Start the Data Links server and DB2 Data Links Manager.
2. Issue the LOAD RESTART command (see “LOAD Command” on page 70).

Links that fail during the load operation are considered to be data integrity violations, and are handled in much the same way as unique index violations. Consequently, a special exception has been defined for loading tables that have one or more DATALINK columns. For additional information, refer to the description of exceptions in the *SQL Reference*.

Chapter 6. Moving Data Between Systems

This chapter describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from DRDA host databases.

DataPropagator (DPROP), another method for moving data between databases in an enterprise, is also described.

The following topics are covered:

- "Moving Data Across Platforms"
- "Moving Data Using the db2move Tool" on page 158
- "Moving Data With DB2 Connect" on page 163
- "Moving Data Between Typed Tables" on page 165
- "Using Replication to Move Data" on page 170.

Moving Data Across Platforms

Compatibility is important when exporting, importing, or loading data across platforms. The following sections describe PC/IXF, delimited ASCII (DEL), and WSF file format considerations when moving data between different operating systems. For more detailed information about the file formats that you can use with the DB2 data movement utilities, see "Appendix C. Export/Import/Load Utility File Formats" on page 179.

PC/IXF File Format

PC/IXF is the recommended file format for transferring data across platforms. PC/IXF files allow the load utility or the import utility to process (normally machine dependent) numeric data in a machine-independent fashion. For example, numeric data is stored and handled differently by Intel and other hardware architectures.

To provide compatibility of PC/IXF files among all products in the DB2 family, the export utility creates files with numeric data in Intel format, and the import utility expects it in this format.

Depending on the hardware platform, DB2 products convert numeric values between Intel and non-Intel formats (using byte reversal) during both export and import operations.

UNIX based implementations of DB2 do not create multiple-part PC/IXF files during export. However, they will allow you to import a multiple-part PC/IXF file that was created by DB2. When importing this type of file, all parts should be in the same directory, otherwise an error is returned.

Single-part PC/IXF files created by UNIX based implementations of the DB2 export utility can be imported by DB2 for OS/2 or DB2 for Windows NT.

Delimited ASCII (DEL) File Format

DEL files have differences based on the operating system on which they were created. The differences are:

- Row separator characters
  - UNIX based text files use a line feed (LF) character.
  - Non-UNIX based text files use a carriage return/line feed (CRLF) sequence.
- End-of-file character
  - UNIX based text files do not have an end-of-file character.
  - Non-UNIX based text files have an end-of-file character (X’1A’).

Since DEL export files are text files, they can be transferred from one operating system to another. File transfer programs can handle operating system-dependant differences if you transfer the files in text mode; the conversion of row separator and end-of-file characters is not performed in binary mode.

Note: If character data fields contain row separator characters, these will also be converted during file transfer. This conversion causes unexpected changes to the data and, for this reason, it is recommended that you do not use DEL export files to move data across platforms. Use the PC/IXF file format instead.

WSF File Format

Numeric data in WSF format files is stored using Intel machine format. This format allows Lotus WSF files to be transferred and used in different Lotus operating environments (for example, in Intel based and UNIX based systems).

As a result of this consistency in internal formats, exported WSF files from DB2 products can be used by Lotus 1-2-3 or Symphony running on a different platform. DB2 products can also import WSF files that were created on different platforms.

Transfer WSF files between operating systems in binary (not text) mode.

Note: Do not use the WSF file format to transfer data between DB2 databases on different platforms, because a loss of data can occur. Use the PC/IXF file format instead.

Moving Data Using the db2move Tool

This tool facilitates the movement of large numbers of tables between DB2 databases located on workstations. The tool queries the system catalog tables for a particular database and compiles a list of all user tables. It then exports these tables in PC/IXF format. The PC/IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another workstation platform and imported or loaded to a DB2 database on that platform.

Authorization

This tool calls the DB2 export, import, and load APIs, depending on the action requested by the user. Therefore, the requesting user ID must have the correct authorization required by those APIs, or the request will fail.

Command Syntax

```
                              ┌─────────────┐
                              ▼             │
►►──db2move──dbname──action───┬───────┬─────┴─────────────────────►◄
                              ├─-tc───┤
                              ├─-tn───┤
                              ├─-io───┤
                              ├─-lo───┤
                              ├─-l────┤
                              ├─-u────┤
                              └─-p────┘
```

Command Parameters

dbname
Name of the database.

action Must be one of: EXPORT, IMPORT, or LOAD.

-tc table-creators. The default is all creators.

This is an EXPORT action only. If specified, only those tables created by the creators listed with this option are exported. If not specified, the default is to use all creators. When specifying multiple creators, each must be separated by commas; no blanks are allowed between creator IDs. The maximum number of creators that can be specified is 10. This option can be used with the "-tn" table-names option to select the tables for export.

An asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-tn table-names. The default is all user tables.

This is an EXPORT action only. If specified, only those tables whose names match exactly those in the specified string are exported. If not specified, the default is to use all user tables. When specifying multiple table names, each must be separated by commas; no blanks are allowed between table names. The maximum number of table names that can be specified is 10. This option can be used with the "-tc" table-creators option to select the tables for export. **db2move** will only export those tables whose names are matched with specified table names and whose creators are matched with specified table creators.

An asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-io import-option. The default is REPLACE_CREATE.

Valid options are: INSERT, INSERT_UPDATE, REPLACE, CREATE, and REPLACE_CREATE.

-lo load-option. The default is INSERT.

Valid options are: INSERT and REPLACE.

-l lobpaths. The default is the current directory.

This option specifies the absolute path names where LOB files are created (as part of EXPORT) or searched for (as part of IMPORT or LOAD). When specifying multiple LOB paths, each must be separated by commas; no blanks are allowed between LOB paths. If the first path runs out of space (during EXPORT), or the files are not found in the path (during IMPORT or LOAD), the second path will be used, and so on.

If the action is EXPORT, and LOB paths are specified, all files in the LOB path directories are deleted, the directories are removed, and new directories are created. If not specified, the current directory is used for the LOB path.

-u userid. The default is the logged on user ID.

Both user ID and password are optional. However, if one is specified, the other must be specified. If the command is run on a client connecting to a remote server, user ID and password should be specified.

-p password. The default is the logged on password.

Both user ID and password are optional. However, if one is specified, the other must be specified. If the command is run on a client connecting to a remote server, user ID and password should be specified.

Examples

- `db2move sample export`

  This will export all tables in the SAMPLE database; default values are used for all options.
- `db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2`

  This will export all tables created by “userid1” or user IDs LIKE “us%rid2”, and with the name “tbname1” or table names LIKE “%tbname2”.
- `db2move sample import -l D:\LOBPATH1,C:\LOBPATH2`

  This example is applicable to OS/2 or the Windows operating system only. The command will import all tables in the SAMPLE database; LOB paths “D:\LOBPATH1” and “C:\LOBPATH2” are to be searched for LOB files.
- `db2move sample load -l /home/userid/lobpath,/tmp`

  This example is applicable to UNIX based systems only. The command will load all tables in the SAMPLE database; both the `/home/userid/lobpath` subdirectory and the `tmp` subdirectory are to be searched for LOB files.
- `db2move sample import -io replace -u userid -p password`

  This will import all tables in the SAMPLE database in REPLACE mode; the specified user ID and password will be used.

Usage Notes

This tool exports, imports, or loads user-created tables. If a database is to be duplicated from one operating system to another operating system, **db2move** facilitates the movement of the tables. It is also necessary to move all other objects associated with the tables, such as: aliases, views, triggers, user-defined functions, and so on. **db2look** (DB2 Statistics Extraction Tool; see the *Command Reference*) can facilitate the movement of some of these objects by extracting the data definition language (DDL) statements from the database.

When export, import, or load APIs are called by **db2move**, the `FileTypeMod` parameter is set to `lobsinfile`. That is, LOB data is kept in separate files from PC/IXF files. There are 26 000 file names available for LOB files.

The LOAD action must be run locally on the machine where the database and the data file reside. When the load API is called by **db2move**, the `CopyTargetList` parameter is set to NULL; that is, no copying is done. If *logretain* is on, the load operation cannot be rolled forward later. The table space where the loaded tables reside is placed in backup pending state, and is not accessible. A full database backup, or a table space backup, is required to take the table space out of backup pending state.

Files Required/Generated When Using EXPORT:

- Input: None.

- Output:

  EXPORT.out The summarized result of the EXPORT action.

  db2move.lst The list of original table names, their corresponding PC/IXF file names (tabnnn.ixf), and message file names (tabnnn.msg). This list, the exported PC/IXF files, and LOB files (tabnnnc.yyy) are used as input to the **db2move** IMPORT or LOAD action.

  tabnnn.ixf The exported PC/IXF file of a specific table.

  tabnnn.msg The export message file of the corresponding table.

  tabnnnc.yyy The exported LOB files of a specific table.

  "nnn" is the table number. "c" is a letter of the alphabet. "yyy" is a number ranging from 001 to 999.

  These files are created only if the table being exported contains LOB data. If created, these LOB files are placed in the "lobpath" directories. There are a total of 26 000 possible names for the LOB files.

  system.msg The message file containing system messages for creating or deleting file or directory commands. This is only used if the action is EXPORT, and a LOB path is specified.

Files Required/Generated When Using IMPORT:

- Input:

  db2move.lst An output file from the EXPORT action.

  tabnnn.ixf An output file from the EXPORT action.

  tabnnnc.yyy An output file from the EXPORT action.
- Output:

  IMPORT.out The summarized result of the IMPORT action.

  tabnnn.msg The import message file of the corresponding table.

Files Required/Generated When Using LOAD:

- Input:

  db2move.lst An output file from the EXPORT action.

  tabnnn.ixf An output file from the EXPORT action.

  tabnnnc.yyy An output file from the EXPORT action.
- Output:

  LOAD.out The summarized result of the LOAD action.

tabnnn.msg The LOAD message file of the corresponding table.

Moving Data With DB2 Connect

If you are working in a complex environment in which you need to move data between a host database system and a workstation, you can use DB2 Connect, the gateway for data transfer from the host to the workstation, as well as the reverse (see Figure 9).

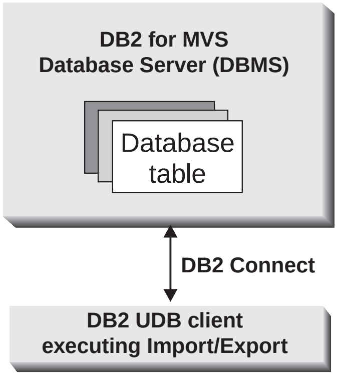


Figure 9. Import/Export through DB2 Connect

The following section discusses exporting and importing data using DB2 Connect.

Using the Export and the Import Utilities

The DB2 export and import utilities allow you to move data from a DRDA server database to a file on the DB2 Connect workstation, and the reverse. You can then use the data with any other application or relational database management system that supports this export or import format. For example, you can export data from DB2 for MVS/ESA into a delimited ASCII file, and then import it into a DB2 for OS/2 database.

You can perform export and import operations from a database client or from the DB2 Connect workstation.

Notes:

1. The data to be exported or imported must comply with the size and data type restrictions that are applicable to both databases.
2. To improve import performance, you can use compound SQL. Specify the `compound` file type modifier in the import utility to group a specified number of SQL statements into a block (see "File Type Modifiers (Import)" on page 49). This may reduce network overhead and improve response time.

3. For detailed information about the syntax of the export and the import utilities, see "EXPORT Command" on page 5, and "IMPORT Command" on page 29.

Moving Data from a Workstation to a DRDA Server

To move data to a DRDA server database:

1. Export the data from a DB2 table to a PC/IXF file.
2. Using the INSERT option, import the PC/IXF file into a compatible table in the DRDA server database.

Moving Data from a DRDA Server to a Workstation

To move data from a DRDA server database:

1. Export the data from the DRDA server database table to a PC/IXF file.
2. Import the PC/IXF file into a DB2 table.

Restrictions

With DB2 Connect, export and import operations must meet the following conditions:

- The file type must be PC/IXF.
- A table with attributes that are compatible with the data must exist before you can import to it. Import through DB2 Connect cannot create a table, because INSERT is the only supported option.
- A commit count interval must not be specified for the import operation.

If any of these conditions is not met, the operation fails, and an error message is returned.

Note: Index definitions are not stored on export or used on import.

Mixed Single-Byte and Double-Byte Data

If you export or import mixed data (columns containing both single-byte and double-byte data), consider the following:

- On systems that store data in EBCDIC (MVS, OS/390, OS/400, VM, and VSE), shift-out and shift-in characters mark the start and the end of double-byte data. When you define column lengths for your database tables, be sure to allow enough room for these characters.
- Variable-length character columns are recommended, unless the column data has a consistent pattern.

Moving Data Between Typed Tables

The DB2 export and import utilities can be used to move data out of, and into, typed tables. Typed tables may be in a hierarchy. Data movement across hierarchies can include:

- Movement from one hierarchy to an identical hierarchy.
- Movement from one hierarchy to a sub-section of a larger hierarchy.
- Movement from a sub-section of a large hierarchy to a separate hierarchy.

The IMPORT CREATE option allows you to create both the table hierarchy and the type hierarchy.

Identification of types in a hierarchy is database dependent. This means that in different databases, the same type has a different identifier. Therefore, when moving data between these databases, a mapping of the same types must be done to ensure that the data is moved correctly.

Before each typed row is written out during an export operation, an identifier is translated into an index value. This index value can be any number from one to the number of relevant types in the hierarchy. Index values are generated by numbering each type when moving through the hierarchy in a specific order. This order is called the *traverse order*. It is the order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy. The traverse order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

One method is to proceed from the top of the hierarchy (or the root table), down the hierarchy (subtables) to the bottom subtable, then back up to its supertable, down to the next "right-most" subtable(s), then back up to next higher supertable, down to its subtables, and so on.

The following figure shows a hierarchy with four valid traverse orders:

- Person, Employee, Manager, Architect, Student.
- Person, Student, Employee, Manager, Architect (this traverse order is marked with the dotted line).
- Person, Employee, Architect, Manager, Student.
- Person, Student, Employee, Architect, Manager.

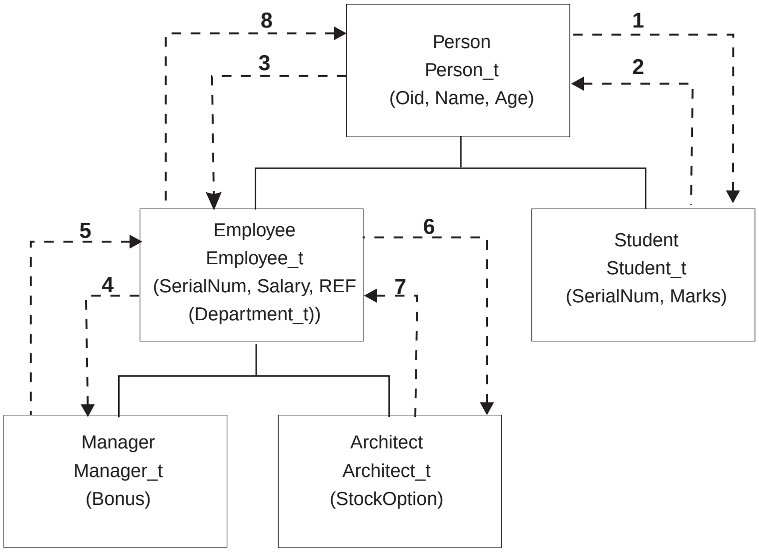


Figure 10.

Traverse Order

There is a default traverse order, in which all relevant types refer to all reachable types in the hierarchy from a given starting point in the hierarchy. The default order includes all tables in the hierarchy, and each table is ordered by the scheme used in the OUTER order predicate. There is also a user-specified traverse order, in which the user defines (in a traverse order list) the relevant types to be used. The same traverse order must be used when invoking the export utility and the import utility.

If you are specifying the traverse order, remember that the subtables must be traversed in PRE-ORDER fashion (that is, each branch in the hierarchy must be traversed to the bottom before a new branch is started).

Default Traverse Order

The default traverse order behaves differently when used with different file formats. Assume identical table hierarchy and type relationships in the following:

Exporting data to the PC/IXF file format creates a record of all relevant types, their definitions, and relevant tables. Export also completes the mapping of an index value to each table. During import, this mapping is used to ensure accurate movement of the data to the target database. When working with the PC/IXF file format, you should use the default traverse order.

With the ASC, DEL, or WSF file format, the order in which the typed rows and the typed tables were created could be different, even though the source and target hierarchies may be structurally identical. This results in time differences that the default traverse order will identify when proceeding through the hierarchies. The creation time of each type determines the order taken through the hierarchy at both the source and the target when using the default traverse order. Ensure that the creation order of each type in both the source and the target hierarchies is identical, and that there is structural identity between the source and the target. If these conditions cannot be met, select a user-specified traverse order.

User-Specified Traverse Order

If you want to control the traverse order through the hierarchies, ensure that the same traverse order is used for both the export and the import utilities. Given:

- An identical definition of subtables in both the source and the target databases
- An identical hierarchical relationship among the subtables in both the source and target databases
- An identical traverse order

the import utility guarantees the accurate movement of data to the target database.

Although you determine the starting point and the path down the hierarchy when defining the traverse order, each branch must be traversed to the end before the next branch in the hierarchy can be started. The export and import utilities look for violations of this condition within the specified traverse order.

Selection During Data Movement

The movement of data from one hierarchical structure of typed tables to another is done through a specific traverse order and the creation of an intermediate flat file. The export utility (in conjunction with the traverse order) controls what is placed in that file. You only need to specify the target table name and the WHERE clause. The export utility uses these selection criteria to create an appropriate intermediate file.

The import utility controls what is placed in the target database. You can specify an attributes list at the end of each subtable name to restrict the attributes that are moved to the target database. If no attributes list is used, all of the columns in each subtable are moved.

The import utility controls the size and the placement of the hierarchy being moved through the CREATE, INTO table-name, UNDER, and AS ROOT TABLE parameters. For detailed information about IMPORT command parameters, see “IMPORT Command” on page 29.

Examples of Moving Data Between Typed Tables

Examples in this section are based on the following hierarchical structure:

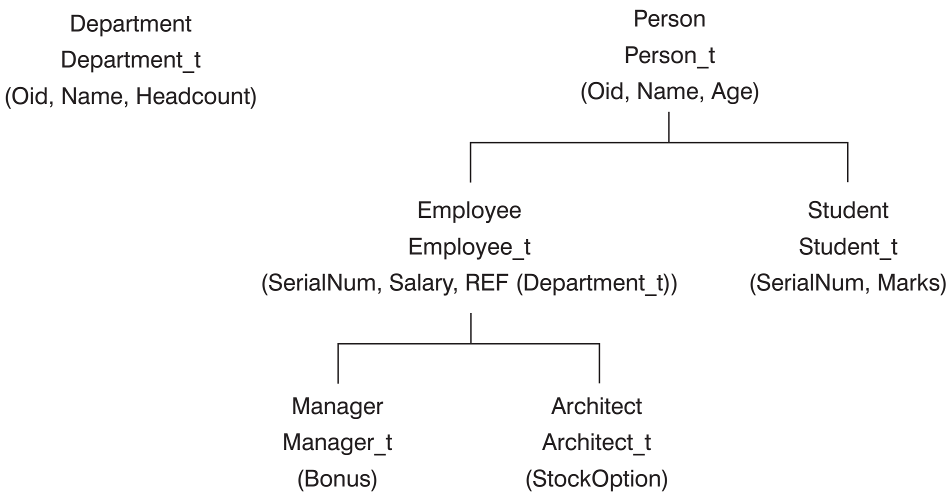


Figure 11.

Example 1

To export an entire hierarchy and then recreate it through an import operation:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.ixf OF IXF CREATE INTO
   HIERARCHY STARTING Person AS ROOT TABLE
```

Each type in the hierarchy is created if it does not exist. If these types already exist, they must have the same definition in the target database as in the source database. An SQL error (SQL20013N) is returned if they are not the same. Since we are creating a new hierarchy, none of the subtables defined in the data file being moved to the target database (`Target_db`) can exist. Each of the tables in the source database hierarchy is created. Data from the source database is imported into the correct subtables of the target database.

Example 2

In a more complex example, we would like to export the entire hierarchy of the source database and import it to the target database. Although we will export all of the data for those people over the age of 20, we will only import selected data to the target database:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.del OF DEL HIERARCHY (Person,
   Employee, Manager, Architect, Student) WHERE Age>=20
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.del OF DEL INSERT INTO (Person,
   Employee(Salary), Architect) IN HIERARCHY (Person, Employee,
   Manager, Architect, Student)
```

The target tables Person, Employee, and Architect must all exist. Data is imported into the Person, Employee, and Architect subtables. That is, we will import:

- All columns in Person into Person.
- All columns in Person plus Salary in Employee into Employee.
- All columns in Person plus Salary in Employee, plus all columns in Architect into Architect.

Columns SerialNum and REF(Employee_t) will not be imported into Employee or its subtables (that is, Architect, which is the only subtable having data imported into it).

Note: Because Architect is a subtable of Employee, and the only import column specified for Employee is Salary, Salary will also be the only Employee-specific column imported into Architect. That is, neither SerialNum nor REF(Employee_t) columns are imported into either Employee or Architect rows.

Data for the Manager and the Student tables is not imported.

Example 3

In this example, we export from a regular table, and import as a single subtable in a hierarchy. The EXPORT command operates on regular (non-typed) tables, so there is no Type_id column in the data file. The modifier no_type_id is used to indicate this, so that the import utility does not expect the first column to be the Type_id column.

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO Student_sub_table.del OF DEL SELECT * FROM
   Regular_Student
DB2 CONNECT TO Target_db
DB2 IMPORT FROM Student_sub_table.del OF DEL METHOD P(1,2,3,5,4)
   MODIFIED BY NO_TYPE_ID INSERT INTO HIERARCHY (Student)
```

In this example, the target table Student must exist. Since Student is a subtable, the modifier no_type_id is used to indicate that there is no Type_id in the first column. However, you must ensure that there is an existing Object_id column, in addition to all of the other attributes that exist in the Student table. Object-id is expected to be the first column in each row imported into the Student table. The METHOD clause reverses the order of the last two attributes.

Using Replication to Move Data

Replication allows you to copy data on a regular basis to multiple remote databases. If you need to have updates to a master database automatically copied to other databases, you can use the replication features of DB2 to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied. The replication features in DB2 are part of a larger IBM solution for replicating data in small and large enterprises—IBM Relational Data Replication (IBM Replication).

The IBM Replication tools are a set of IBM DataPropagator (DPROP) programs and DB2 Universal Database tools that copy data between distributed relational database management systems:

- Between DB2 Universal Database platforms.
- Between DB2 Universal Database platforms and host databases supporting Distributed Relational Database Architecture (DRDA) connectivity.
- Between host databases that support DRDA connectivity.

You can use the IBM Replication tools to define, synchronize, automate, and manage copy operations from a single control point for data across your enterprise. The replication tools in DB2 Universal Database offer replication between relational databases only.

Replication allows you to give end users and applications access to production data without putting extra load on the production database. You can copy the data to a database that is local to a user or an application, rather than have them access the data remotely. A typical replication scenario involves a source table with copies in one or more remote databases; for example, a central bank and its local branches. At predetermined times, automatic updates of the DB2 databases takes place, and all changes to the source database are copied to the target database tables.

The replication tools allow you to customize the copy table structure. You can use SQL when copying to the target database to enhance the data being copied. You can produce read-only copies that duplicate the source table, capture data at a specified point in time, provide a history of changes, or stage data to be copied to additional target tables. Moreover, you can create read-write copies that can be updated by end users or applications, and then have the changes replicated back to the master table. You can replicate views of source tables, or views of copies. Event-driven replication is also possible.

The replication tools currently support DB2 on MVS/ESA, AS/400, AIX, OS/2, VM and VSE, Windows NT, HP, and the Solaris operating environment. You can also replicate to non-IBM databases, such as Oracle, Microsoft SQL Server, and Lotus Notes.

The IBM Replication Tools by Component

There are two components of the IBM Replication tools solution: IBM DPROP Capture, and IBM DPROP Apply. You can set up these components using the DB2 Control Center. The operation and monitoring of these components happens outside of the Control Center.

The IBM DPROP Capture program captures changes to the source tables. A source table can be:

- An external table containing SQL data from a file system or a nonrelational database manager loaded outside DPROP.
- An existing table in the database.
- A table that has previously been updated by the IBM DPROP Apply program, which allows changes to be copied back to the source, or to other target tables.

The changes are copied into a change data table, where they are stored until the target system is ready to copy them. The Apply program then takes the changes from the change data table, and copies them to the target tables.

Use the Control Center to:

- Set up the replication environment.
- Define source and target tables.
- Specify the timing of automated copying.
- Specify SQL enhancements to the data.
- Define relationships between the source and the target tables.

For more information, see the *Replication Guide and Reference*.

Appendix A. How to Read the Syntax Diagrams

A syntax diagram shows how a command should be specified so that the operating system can correctly interpret what is typed.

Read a syntax diagram from left to right, and from top to bottom, following the horizontal line (the main path). If the line ends with an arrowhead, the command syntax is continued, and the next line starts with an arrowhead. A vertical bar marks the end of the command syntax.

When typing information from a syntax diagram, be sure to include punctuation, such as quotation marks and equal signs.

Parameters are classified as keywords or variables:

- Keywords represent constants, and are shown in uppercase letters; at the command prompt, however, keywords can be entered in upper, lower, or mixed case. A command name is an example of a keyword.
- Variables represent names or values that are supplied by the user, and are shown in lowercase letters; at the command prompt, however, variables can be entered in upper, lower, or mixed case, unless case restrictions are explicitly stated. A file name is an example of a variable.

A parameter can be a combination of a keyword and a variable.

Required parameters are displayed on the main path:

```
►►──COMMAND─required parameter─────────────────────────────────────────────►◄
```

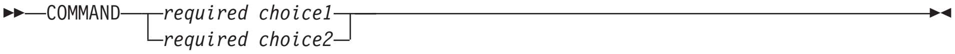
Optional parameters are displayed below the main path:

```
►►──COMMAND──┬────────────────────┬────────────────────────────────────────►◄
             └─optional parameter─┘
```

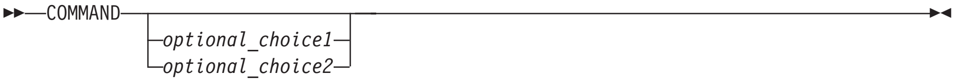
A parameter's default value is displayed above the path:

```
►►──COMMAND──┬──────────────────────────┬──────────────────────────────────►◄
             │              ┌─VALUE1──┐ │
             └─OPTPARM──────┼─VALUE2──┼─┘
                            ├─VALUE3──┤
                            └─VALUE4──┘
```

A stack of parameters, with the first parameter displayed on the main path, indicates that one of the parameters must be selected:

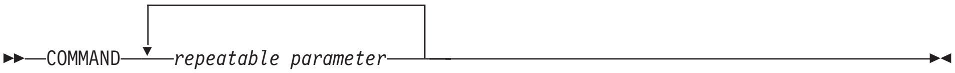


A stack of parameters, with the first parameter displayed below the main path, indicates that one of the parameters can be selected:

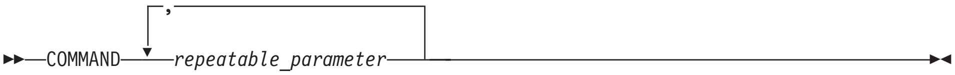


An arrow returning to the left, above the path, indicates that items can be repeated in accordance with the following conventions:

- If the arrow is uninterrupted, the item can be repeated in a list with the items separated by blank spaces:

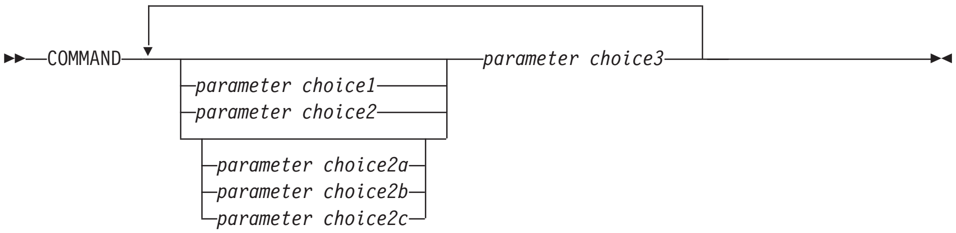


- If the arrow contains a comma, the item can be repeated in a list with the items separated by commas:

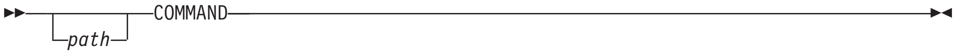


Items from parameter stacks can be repeated in accordance with the stack conventions for required and optional parameters discussed previously.

Some syntax diagrams contain parameter stacks within other parameter stacks. Items from stacks can only be repeated in accordance with the conventions discussed previously. That is, if an inner stack does not have a repeat arrow above it, but an outer stack does, only one parameter from the inner stack can be chosen and combined with any parameter from the outer stack, and that combination can be repeated. For example, the following diagram shows that one could combine parameter *choice2a* with parameter *choice2*, and then repeat that combination again (*choice2* plus *choice2a*):

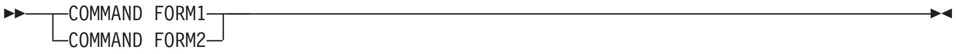


Some commands are preceded by an optional path parameter:



If this parameter is not supplied, the system searches the current directory for the command. If it cannot find the command, the system continues searching for the command in all the directories on the paths listed in the `.profile`.

Some commands have syntactical variants that are functionally equivalent:



Appendix B. Differences Between the Import and the Load Utility

The following table summarizes the important differences between the DB2 load and import utilities.

Import Utility	Load Utility
Slow when moving large amounts of data.	Faster than the import utility when moving large amounts of data, because the load utility writes formatted pages directly into the database.
Limited exploitation of intra-partition parallelism.	Exploitation of intra-partition parallelism. Typically, this requires symmetric multiprocessor (SMP) machines.
No FASTPARSE support.	FASTPARSE support, providing reduced data checking of user-supplied data.
No CODEPAGE support.	CODEPAGE support, converting character data (and numeric data specified in characters) from this code page to the database code page during the load operation.
Supports hierarchical data.	Does not support hierarchical data.
Creation of tables, hierarchies, and indexes supported with PC/IXF format.	Tables and indexes must exist.
No support for importing into summary tables.	Support for loading into summary tables.
WSF format is supported.	WSF format is not supported.
No BINARYNUMERICS support.	BINARYNUMERICS support.
No PACKEDDECIMAL support.	PACKEDDECIMAL support.
Supports import into tables and views.	Supports loading into tables only.
The table spaces in which the table and its indexes reside are online for the duration of the import operation.	The table spaces in which the table and its indexes reside are offline for the duration of the load operation.
All rows are logged.	Minimal logging is performed.
Trigger support.	No trigger support.

Differences Between the Import and the Load Utility

Import Utility	**Load Utility**
If an import operation is interrupted, and a *commitcount* was specified, the table is usable and will contain the rows that were loaded up to the last COMMIT. The user can restart the import operation, or accept the table as is.	If a load operation is interrupted, and a *savecount* was specified, the table remains in load pending state and cannot be used until the load operation is restarted, a load terminate operation is invoked, or until the table space is restored from a backup image created some time before the attempted load operation.
Space required is approximately equivalent to the size of the largest index plus 10%. This space is obtained from the temporary table spaces within the database.	Space required is approximately equivalent to the sum of the size of all indexes defined on the table, and can be as much as twice this size. This space is obtained from temporary space within the database.
All constraints are validated during an import operation.	Uniqueness is verified during a load operation, but all other constraints must be checked using the SET INTEGRITY statement.
The key values are inserted into the index one at a time during an import operation.	The key values are sorted and the index is built after the data has been loaded.
If updated statistics are required, the runstats utility must be run after an import operation.	Statistics can be gathered during the load operation if all the data in the table is being replaced.
You can import into a host database through DB2 Connect.	You cannot load into a host database.
Import files must reside on the node from which the import utility is invoked.	Load files or pipes must reside on the node that contains the database.
A backup image is not required. Because the import utility uses SQL inserts, DB2 logs the activity, and no backups are required to recover these operations in case of failure.	A backup image can be created during the load operation.

Appendix C. Export/Import/Load Utility File Formats

Four operating system file formats supported by the DB2 export, import, and load utilities are described:

DEL Delimited ASCII, for data exchange among a wide variety of database managers and file managers. This common approach to storing data uses special character delimiters to separate column values.

ASC Non-delimited ASCII, for importing or loading data from other applications that create flat text files with aligned column data.

PC/IXF
PC version of the Integrated Exchange Format (IXF), the preferred method for data exchange within the database manager. PC/IXF is a structured description of a database table that contains an external representation of the internal table.

WSF Work-sheet format, for data exchange with products such as Lotus 1-2-3 and Symphony. The load utility does not support this file format.

When using DEL, WSF, or ASC data file formats, define the table, including its column names and data types, before importing the file. The data types in the operating system file fields are converted into the corresponding type of data in the database table. The import utility accepts data with minor incompatibility problems, including character data imported with possible padding or truncation, and numeric data imported into different types of numeric fields.

When using the PC/IXF data file format, the table does not need to exist before beginning the import operation. User-defined distinct types (UDTs) are not made part of the new table column types; instead, the base type is used. Similarly, when exporting to the PC/IXF data file format, UDTs are stored as base data types in the PC/IXF file.

Delimited ASCII (DEL) File Format

A Delimited ASCII (DEL) file is a sequential ASCII file with row and column delimiters. Each DEL file is a stream of ASCII characters consisting of cell values ordered by row, and then by column. Rows in the data stream are separated by row delimiters; within each row, individual cell values are separated by column delimiters.

Delimited ASCII (DEL) File Format

The following table describes the format of DEL files that can be imported, or that can be generated as the result of an export action.

```
DEL file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data || Optional row delimiter

Row i data ::= Cell value(i,1) || Column delimiter ||
               Cell value(i,2) || Column delimiter ||
               .
               .
               .
               Cell value(i,m)

Row delimiter ::= ASCII line feed sequence[a]

Column delimiter ::= Default value ASCII comma (,)[b]

Cell value(i,j) ::= Leading spaces
                 || ASCII representation of a numeric value
                    (integer, decimal, or float)
                 || Delimited character string
                 || Non-delimited character string
                 || Trailing spaces

Non-delimited character string ::= A set of any characters except a row delimiter
                                   or a column delimiter

Delimited character string ::= A character string delimiter ||
                               An extended character string ||
                               A character string delimiter ||
                               Trailing garbage

Trailing garbage ::= A set of any characters except a row delimiter
                     or a column delimiter

Character string delimiter ::= Default value ASCII double quotation
                               marks (")[c]

extended character string ::= || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the NODOUBLEDEL
                                 modifier is specified
                              || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the character string
                                 is not part of two consecutive
                                 character string delimiters
                              || A set of any characters except a
                                 character string delimiter if the
                                 character string delimiter is not
                                 part of two consecutive character
```

```
                                              string delimiters, and the DELPRIORITYCHAR
                                              modifier is specified

End-of-file character ::= Hex '1A' (OS/2 or the Windows operating system only)

ASCII representation of a numeric value[d] ::= Optional sign '+' or '-'
      || 1 to 31 decimal digits with an optional decimal point before,
         after, or between two digits
      || Optional exponent

Exponent ::= Character 'E' or 'e'
      || Optional sign '+' or '-'
      || 1 to 3 decimal digits with no decimal point

Decimal digit ::= Any one of the characters '0', '1', ... '9'

Decimal point ::= Default value ASCII period (.)[e]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on OS/2 or the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the CODEPAGE option on the LOAD command).
- [b] The column delimiter can be specified with the COLDEL option.
- [c] The character string delimiter can be specified with the CHARDEL option.

  Note: The default priority of delimiters is:

  1. Record delimiter
  2. Character delimiter
  3. Column delimiter

  See also the description of the delprioritychar modifier in Table 8 on page 104.
- [d] If the ASCII representation of a numeric value contains an exponent, it is a FLOAT constant. If it has a decimal point but no exponent, it is a DECIMAL constant. If it has no decimal point and no exponent, it is an INTEGER constant.
- [e] The decimal point character can be specified with the DECPT option.

Sample DEL File

Following is an example of a DEL file. Each line ends with a line feed sequence (on OS/2 or the Windows operating system, each line ends with a carriage return/line feed sequence).

```
"Smith, Bob",4973,15.46
"Jones, Bill",12345,16.34
"Williams, Sam",452,193.78
```

The following example illustrates the use of non-delimited character strings. The column delimiter has been changed to a semicolon, because the character data contains a comma.

```
Smith, Bob;4973;15.46
Jones, Bill;12345;16.34
Williams, Sam;452;193.78
```

Notes:

1. A space (X'20') is never a valid delimiter.
2. Spaces that precede the first character, or that follow the last character of a cell value, are discarded during import. Spaces that are embedded in a cell value are not discarded.
3. A period (.) is not a valid character string delimiter, because it conflicts with periods in time stamp values.
4. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive.
5. For DEL data specified in an EBCDIC code page, the delimiters may not coincide with the shift-in and shift-out DBCS characters.
6. On OS/2 or the Windows operating system, the first occurrence of an end-of-file character (X'1A') that is not within character delimiters indicates the end-of-file. Any subsequent data is not imported.
7. A null value is indicated by the absence of a cell value where one would normally occur, or by a string of spaces.
8. Since some products restrict character fields to 254 or 255 bytes, the export utility generates a warning message whenever a character column of maximum length greater than 254 bytes is selected for export. The import utility accommodates fields that are as long as the longest LONG VARCHAR and LONG VARGRAPHIC columns.

DEL Data Type Descriptions

Table 9. Acceptable Data Type Forms for the DEL File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	An INTEGER constant in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.	ASCII representation of a numeric value in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal and float numbers are truncated to integer values.
BLOB, CLOB	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.
BLOB_FILE, CLOB_FILE	The character data for each BLOB/CLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.
CHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated or padded with spaces (X'20'), if necessary, to match the width of the database column.
DATE	*yyyymmdd* (year month day) with no character delimiters. For example: 19931029 Alternatively, the DATESISO option can be used to specify that all date values are to be exported in ISO format.	A delimited or non-delimited character string containing a date value in an ISO format consistent with the country code of the target database, or a non-delimited character string of the form *yyyymmdd*.
DBCLOB (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.

Table 9. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
DBCLOB_FILE (DBCS only)	The character data for each DBCLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.
DECIMAL	A DECIMAL constant with the precision and scale of the field being exported. The DECPLUSBLANK option can be used to specify that positive decimal values are to be prefixed with a blank space instead of a plus sign (+).	ASCII representation of a numeric value that does not overflow the range of the database column into which the field is being imported. If the input value has more digits after the decimal point than can be accommodated by the database column, the excess digits are truncated.
FLOAT(long)	A FLOAT constant in the range -10E307 to 10E307.	ASCII representation of a numeric value in the range -10E307 to 10E307.
GRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated or padded with double-byte spaces (for example, X'8140'), if necessary, to match the width of the database column.
INTEGER	An INTEGER constant in the range -2 147 483 648 to 2 147 483 647.	ASCII representation of a numeric value in the range -2 147 483 648 to 2 147 483 647. Decimal and float numbers are truncated to integer values.
LONG VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.

Table 9. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
LONG VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.
SMALLINT	An INTEGER constant in the range -32 768 to 32 767.	ASCII representation of a numeric value in the range -32 768 to 32 767. Decimal and float numbers are truncated to integer values.
TIME	*hh.mm.ss* (hour minutes seconds). A time value in ISO format enclosed by character delimiters. For example: "09.39.43"	A delimited or non-delimited character string containing a time value in a format consistent with the country code of the target database.
TIMESTAMP	*yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). A character string representing a date and time enclosed by character delimiters.	A delimited or non-delimited character string containing a time stamp value acceptable for storage in a database.
VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated, if necessary, to match the maximum width of the database column.
VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated, if necessary, to match the maximum width of the database column.

Non-delimited ASCII (ASC) File Format

A non-delimited ASCII (ASC) file is a sequential ASCII file with row delimiters. It can be used for data exchange with any ASCII product that has a columnar format for data, including word processors. Each ASC file is a stream of ASCII characters consisting of data values ordered by row and column. Rows in the data stream are separated by row delimiters. Each column within a row is defined by a beginning-ending location pair (specified by IMPORT parameters). Each pair represents locations within a row specified as byte positions. The first position within a row is byte position 1. The first element of each location pair is the byte on which the column begins, and the second element of each location pair is the byte on which the column ends. The columns may overlap. Every row in an ASC file has the same column definition.

An ASC file is defined by:

```
ASC file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data
```

```
Row i data ::= ASCII characters || Row delimiter
```

```
Row Delimiter ::= ASCII line feed sequence[a]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on OS/2 or the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the CODEPAGE option on the LOAD command). The record delimiter is never interpreted to be part of a field of data.

Sample ASC File

Following is an example of an ASC file. Each line ends with a line feed sequence (on OS/2 or the Windows operating system, each line ends with a carriage return/line feed sequence).

```
Smith, Bob       4973     15.46
Jones, Suzanne   12345    16.34
Williams, Sam    452123  193.78
```

Notes:

1. ASC files are assumed not to contain column names.

2. Character strings are *not* enclosed by delimiters. The data type of a column in the ASC file is determined by the data type of the target column in the database table.
3. A NULL is imported into a nullable database column if:
   - A field of blanks is targeted for a numeric, DATE, TIME, or TIMESTAMP database column
   - A field with no beginning and ending location pairs is specified
   - A location pair with beginning and ending locations equal to zero is specified
   - A row of data is too short to contain a valid value for the target column
   - The NULL INDICATORS load option is used, and an `N` (or other value specified by the user) is found in the null indicator column.
4. If the target column is not nullable, an attempt to import a field of blanks into a numeric, DATE, TIME, or TIMESTAMP column causes the row to be rejected.
5. If the input data is not compatible with the target column, and that column is nullable, a null is imported or the row is rejected, depending on where the error is detected. If the column is not nullable, the row is rejected. Messages are written to the message file, specifying incompatibilities that are found.

ASC Data Type Descriptions

Table 10. Acceptable Data Type Forms for the ASC File Format

Data Type	Form Acceptable to the Import Utility
BIGINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
BLOB/CLOB	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.

Table 10. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
BLOB_FILE, CLOB_FILE, DBCLOB_FILE (DBCS only)	A delimited or non-delimited name of the file that holds the data.
CHAR	A string of characters. The character string is truncated or padded with spaces on the right, if necessary, to match the width of the target column.
DATE	A character string representing a date value in a format consistent with the country code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a date.
DBCLOB (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.
DECIMAL	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the database column into which they are being imported. If the input value has more digits after the decimal point than the scale of the database column, the excess digits are truncated. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
FLOAT(long)	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are valid. A comma, period, or colon is considered to be a decimal point. An uppercase or lowercase E is accepted as the beginning of the exponent of a FLOAT constant. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.

Table 10. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
GRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated or padded with double-byte spaces (0x8140) on the right, if necessary, to match the maximum length of the target column.
INTEGER	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
LONG VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
LONG VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.
SMALLINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
TIME	A character string representing a time value in a format consistent with the country code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time.

Table 10. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
TIMESTAMP	A character string representing a time stamp value acceptable for storage in a database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time stamp.
VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.

PC Version of IXF File Format

The PC version of IXF (PC/IXF) file format is a database manager adaptation of the Integration Exchange Format (IXF) data interchange architecture. The IXF architecture was specifically designed to enable the exchange of relational database structures and data. The PC/IXF architecture allows the database manager to export a database without having to anticipate the requirements and idiosyncrasies of a receiving product. Similarly, a product importing a PC/IXF file need only understand the PC/IXF architecture; the characteristics of the product which exported the file are not relevant. The PC/IXF file architecture maintains the independence of both the exporting and the importing database systems.

The IXF architecture is a generic relational database exchange format that supports a rich set of relational data types, including some types that may not be supported by specific relational database products. The PC/IXF file format preserves this flexibility; for example, the PC/IXF architecture supports both single-byte character string (SBCS) and double-byte character string (DBCS) data types. Not all implementations support all PC/IXF data types; however, even restricted implementations provide for the detection and disposition of unsupported data types during import.

In general, a PC/IXF file consists of an unbroken sequence of variable-length records. The file contains the following record types in the order shown:

- One header record of record type H
- One table record of record type T

- Multiple column descriptor records of record type C (one record for each column in the table)
- Multiple data records of record type D (each row in the table is represented by one or more D records).

A PC/IXF file may also contain application records of record type A, anywhere after the H record. These records are permitted in PC/IXF files to enable an application to include additional data, not defined by the PC/IXF format, in a PC/IXF file. A records are ignored by any program reading a PC/IXF file that does not have particular knowledge about the data format and content implied by the application identifier in the A record.

Every record in a PC/IXF file begins with a record length indicator. This is a 6-byte right justified character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Programs reading PC/IXF files should use these record lengths to locate the end of the current record and the beginning of the next record. H, T, and C records must be sufficiently large to include all of their defined fields, and, of course, their record length fields must agree with their actual lengths. However, if extra data (for example, a *new* field), is added to the end of one of these records, pre-existing programs reading PC/IXF files should ignore the extra data, and generate no more than a warning message. Programs writing PC/IXF files, however, should write H, T and C records that are the precise length needed to contain all of the defined fields.

PC/IXF file records are composed of fields which contain character data. The import and export utilities interpret this character data using the CPGID of the target database, with two exceptions:

- The IXFADATA field of A records.

  The code page environment of character data contained in an IXFADATA field is established by the application which creates and processes a particular A record; that is, the environment varies by implementation.
- The IXFDCOLS field of D records.

  The code page environment of character data contained in an IXFDCOLS field is a function of information contained in the C record which defines a particular column and its data.

Numeric fields in H, T, and C records, and in the prefix portion of D and A records should be right justified single-byte character representations of integer values, filled with leading zeros or blanks. A value of zero should be indicated with at least one (right justified) zero character, not blanks. Whenever one of these numeric fields is not used, for example IXFCLENG, where the length is implied by the data type, it should be filled with blanks. These numeric fields are:

```
   IXFHRECL, IXFTRECL, IXFCRECL, IXFDRECL, IXFARECL,
   IXFHHCNT, IXFHSBCP, IXFHDBCP, IXFTCCNT, IXFTNAML,
   IXFCLENG, IXFCDRID, IXFCPOSN, IXFCNAML, IXFCTYPE,
   IXFCSBCP, IXFCDBCP, IXFCNDIM, IXFCDSIZ, IXFDRID
```

Note: The database manager PC/IXF file format is not identical to the System/370 IXF format (see “Differences between Version 1 PC/IXF and Version 0 System/370 IXF” on page 227).

PC/IXF Record Types

There are five PC/IXF record types:

- header
- table
- column descriptor
- data
- application

Each PC/IXF record type is defined as a sequence of fields; these fields are required, and must appear in the order shown.

HEADER RECORD

FIELD NAME	LENGTH	TYPE	COMMENTS
IXFHRECL	06-BYTE	CHARACTER	record length
IXFHRECT	01-BYTE	CHARACTER	record type = 'H'
IXFHID	03-BYTE	CHARACTER	IXF identifier
IXFHVERS	04-BYTE	CHARACTER	IXF version
IXFHPROD	12-BYTE	CHARACTER	product
IXFHDATE	08-BYTE	CHARACTER	date written
IXFHTIME	06-BYTE	CHARACTER	time written
IXFHHCNT	05-BYTE	CHARACTER	heading record count
IXFHSBCP	05-BYTE	CHARACTER	single byte code page
IXFHDBCP	05-BYTE	CHARACTER	double byte code page
IXFHFIL1	02-BYTE	CHARACTER	reserved

The following fields are contained in the header record:

IXFHRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The H record must be sufficiently long to include all of its defined fields.

IXFHRECT
The IXF record type, which is set to H for this record.

IXFHID
The file format identifier, which is set to IXF for this file.

IXFHVERS
The PC/IXF format level used when the file was created, which is set to '0001'.

IXFHPROD
A field that can be used by the program creating the file to identify itself. If this field is filled in, the first six bytes are used to identify the product creating the file, and the last six bytes are used to indicate the version or release of the creating product. The database manager uses this field to signal the existence of database manager-specific data.

IXFHDATE
The date on which the file was written, in the form *yyyymmdd*.

IXFHTIME
The time at which the file was written, in the form *hhmmss*. This field is optional and can be left blank.

IXFHHCNT
The number of H, T, and C records in this file that precede the first data record. A records are not included in this count.

IXFHSBCP
Single-byte code page field, containing a single-byte character representation of a SBCS CPGID or '00000'.

The export utility sets this field equal to the SBCS CPGID of the exported database table. For example, if the table SBCS CPGID is 850, this field contains '00850'.

IXFHDBCP
Double-byte code page field, containing a single-byte character representation of a DBCS CPGID or '00000'.

The export utility sets this field equal to the DBCS CPGID of the exported database table. For example, if the table DBCS CPGID is 301, this field contains '00301'.

IXFHFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

```
TABLE RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFTRECL       06-BYTE   CHARACTER   record length
   IXFTRECT       01-BYTE   CHARACTER   record type = 'T'
   IXFTNAML       03-BYTE   CHARACTER   name length
   IXFTNAME       128-BYTE  CHARACTER   name of data
   IXFTQUAL       128-BYTE  CHARACTER   qualifier
   IXFTSRC        12-BYTE   CHARACTER   data source
   IXFTDATA       01-BYTE   CHARACTER   data convention = 'C'
   IXFTFORM       01-BYTE   CHARACTER   data format = 'M'
   IXFTMFRM       05-BYTE   CHARACTER   machine format='PC'
   IXFTLOC        01-BYTE   CHARACTER   data location = 'I'
   IXFTCCNT       05-BYTE   CHARACTER   'C' record count
   IXFTFIL1       02-BYTE   CHARACTER   reserved
   IXFTDESC       30-BYTE   CHARACTER   data description
```

The following fields are contained in the table record:

IXFTRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The T record must be sufficiently long to include all of its defined fields.

IXFTRECT
The IXF record type, which is set to T for this record.

IXFTNAML
The length, in bytes, of the table name in the IXFTNAME field.

IXFTNAME
The name of the table. If each file has only one table, this is an informational field only. The database manager does not use this field when importing data. When writing a PC/IXF file, the database manager writes the DOS file name (and possibly path information) to this field.

IXFTQUAL
Table name qualifier, which identifies the creator of a table in a relational system. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file may print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTSRC
Used to indicate the original source of the data. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading

a file may print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTDATA
Convention used to describe the data. This field must be set to `C` for import and export, indicating that individual column attributes are described in the following column descriptor (C) records, and that data follows PC/IXF conventions.

IXFTFORM
Convention used to store numeric data. This field must be set to `M`, indicating that numeric data in the data (D) records is stored in the machine (internal) format specified by the IXFTMFRM field.

IXFTMFRM
The format of any machine data in the PC/IXF file. The database manager will only read or write files if this field is set to `PC`*bbb*, where *b* represents a blank, and PC specifies that data in the PC/IXF file is in IBM PC machine format.

IXFTLOC
The location of the data. The database manager only supports a value of I, meaning the data is internal to this file.

IXFTCCNT
The number of C records in this table. It is a right-justified character representation of an integer value.

IXFTFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

IXFTDESC
Descriptive data about the table. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file may print or display this field, or store it in an informational field, but no computations should depend on the content of this field. This field contains `NOT NULL WITH DEFAULT` if the column was not null with default, and the table name came from a workstation database.

```
COLUMN DESCRIPTOR RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFCRECL       06-BYTE   CHARACTER   record length
   IXFCRECT       01-BYTE   CHARACTER   record type = 'C'
   IXFCNAML       03-BYTE   CHARACTER   column name length
   IXFCNAME       30-BYTE   CHARACTER   column name
   IXFCNULL       01-BYTE   CHARACTER   column allows nulls
   IXFCSLCT       01-BYTE   CHARACTER   column selected flag
   IXFCKEY        01-BYTE   CHARACTER   key column flag
   IXFCCLAS       01-BYTE   CHARACTER   data class
   IXFCTYPE       03-BYTE   CHARACTER   data type
   IXFCSBCP       05-BYTE   CHARACTER   single byte code page
   IXFCDBCP       05-BYTE   CHARACTER   double byte code page
   IXFCLENG       05-BYTE   CHARACTER   column data length
   IXFCDRID       03-BYTE   CHARACTER   'D' record identifier
   IXFCPOSN       06-BYTE   CHARACTER   column position
   IXFCDESC       30-BYTE   CHARACTER   column description
   IXFCNDIM       02-BYTE   CHARACTER   number of dimensions
   IXFCDSIZ       varying   CHARACTER   size of each dimension
```

The following fields are contained in column descriptor records:

IXFCRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The C record must be sufficiently long to include all of its defined fields.

IXFCRECT
The IXF record type, which is set to C for this record.

IXFCNAML
The length, in bytes, of the column name in the IXFCNAME field.

IXFCNAME
The name of the column.

IXFCNULL
Specifies if nulls are permitted in this column. Valid settings are Y or N.

IXFCSLCT
An obsolete field whose intended purpose was to allow selection of a subset of columns in the data. Programs writing PC/IXF files should always store a Y in this field. Programs reading PC/IXF files should ignore the field.

IXFCKEY
The key indicator. If the value of this field is Y, the column is a key column; if the value is N, the column is not a key column. The

database manager does not use this field. It ignores the field when importing data, and sets it to N when generating an export file.

IXFCCLAS

The class of data types to be used in the IXFCTYPE field. The database manager only supports relational types (R).

IXFCTYPE

The data type for the column. For more information about data types, see "PC/IXF Data Types" on page 201.

IXFCSBCP

Contains a single-byte character representation of a SBCS CPGID. This field specifies the CPGID for single-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should normally contain a nonzero value equal to that of the IXFHSBCP field in the H record; however, other values are permitted. If this value is zero, the column is interpreted to contain bit string data.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to the value of the IXFHSBCP field by the export utility, and ignored by the import utility.
- For a graphic column, this field must be zero.

See also Table 12 on page 207.

IXFCDBCP

Contains a single-byte character representation of a DBCS CPGID. This field specifies the CPGID for double-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should either be zero, or contain a value equal to that of the IXFHDBCP field in the H record; however, other values are permitted. If the value in the IXFCSBCP field is zero, the value in this field must be zero.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.

- For a graphic column, this field must have a value equal to the value of the IXFHDBCP field.

See also Table 12 on page 207.

IXFCLENG

Provides information about the size of the column being described. For some data types, this field is unused, and should contain blanks. For other data types, this field contains the right-justified character representation of an integer specifying the column length. For yet other data types, this field is divided into two subfields: 3 bytes for precision, and 2 bytes for scale; both of these subfields are right-justified character representations of integers.

IXFCDRID

The D record identifier. This field contains the right-justified character representation of an integer value. Several D records can be used to contain each row of data in the PC/IXF file. This field specifies which D record (of the several D records contributing to a row of data) contains the data for the column. A value of one (for example, 001) indicates that the data for a column is in the first D record in a row of data. The first C record must have an IXFCDRID value of one. All subsequent C records must have an IXFCDRID value equal to the value in the preceding C record, or one higher.

IXFCPOSN

The value in this field is used to locate the data for the column within one of the D records representing a row of table data. It is the starting position of the data for this column within the IXFDCOLS field of the D record. If the column is nullable, IXFCPOSN points to the null indicator; otherwise, it points to the data itself. If a column contains varying length data, the data itself begins with the current length indicator. The IXFCPOSN value for the first byte in the IXFDCOLS field of the D record is one (not zero). If a column is in a new D record, the value of IXFCPOSN should be one; otherwise, IXFCPOSN values should increase from column to column to such a degree that the data values do not overlap.

IXFCDESC

Descriptive information about the column. This is an informational field only. If a program writing to a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file may print or display this field, or store it in an informational field, but no computations should depend on the content of this field. If

- the column is not null with default
- the table resides in a workstation database

- the select statement on the export is of the form `select * from table`

the export utility will put `NOT NULL WITH DEFAULT` in this field, and when the import utility creates a new table with this file, it will create it as not null with default.

IXFCNDIM
The number of dimensions in the column. Arrays are not supported in this version of PC/IXF. This field must therefore contain a character representation of a zero integer value.

IXFCDSIZ
The size or range of each dimension. The length of this field is five bytes per dimension. Since arrays are not supported (that is, the number of dimensions must be zero), this field has zero length, and does not actually exist.

```
DATA RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFDRECL       06-BYTE   CHARACTER   record length
   IXFDRECT       01-BYTE   CHARACTER   record type = 'D'
   IXFDRID        03-BYTE   CHARACTER   'D' record identifier
   IXFDFIL1       04-BYTE   CHARACTER   reserved
   IXFDCOLS       varying   variable    columnar data
```

The following fields are contained in the data records:

IXFDRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each D record must be sufficiently long to include all significant data for the current occurrence of the last data column stored in the record.

IXFDRECT
The IXF record type, which is set to D for this record, indicating that it contains data values for the table.

IXFDRID
The record identifier, which identifies a particular D record within the sequence of several D records contributing to a row of data. For the first D record in a row of data, this field has a value of one; for the second D record in a row of data, this field has a value of two, and so on. In each row of data, all the D record identifiers called out in the C records must actually exist.

IXFDFIL1
Spare field set to four blanks to match reserved fields, and hold a place for a possible shift-out character, in host IXF files.

IXFDCOLS
The area for columnar data. The data area of a data record (D record) is composed of one or more column entries. There is one column entry for each column descriptor record, which has the same D record identifier as the D record. In the D record, the starting position of the column entries is indicated by the IXFCPOSN value in the C records.

The format of the column entry data depends on whether or not the column is nullable:

- If the column is nullable (the IXFCNULL field is set to Y), the column entry data includes a null indicator. If the column is not null, the indicator is followed by data type-specific information, including the actual database value. The null indicator is a two-byte value set to x'0000' for not null, and x'FFFF' for null.
- If the column is not nullable, the column entry data includes only data type-specific information, including the actual database value.

For varying-length data types, the data type-specific information includes a current length indicator. The current length indicators are 2-byte integers in a form specified by the IXFTMFRM field.

The length of the data area of a D record may not exceed 32 771 bytes.

```
APPLICATION RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFARECL       06-BYTE   CHARACTER   record length
   IXFARECT       01-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       12-BYTE   CHARACTER   application identifier
   IXFADATA       varying   variable    application-specific data
```

The following fields are contained in application records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs

which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies the application creating the A record. PC/IXF files created by the database manager may have A records with the first 6 characters of this field set to a constant identifying the database manager, and the last 6 characters identifying the release or version of the database manager or another application writing the A record.

IXFADATA
This field contains application dependent supplemental data, whose form and content are known only to the program creating the A record, and to other applications which are likely to process the A record.

PC/IXF Data Types

Table 11. PC/IXF Data Types

Name	IXFCTYPE Value	Description
BIGINT	492	An 8-byte integer in the form specified by IXFTMFRM. It represents a whole number between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807. IXFCSBCP and IXFCDBCP are not significant , and should be zero. IXFCLENG is not used, and should contain blanks.
BLOB, CLOB	404, 408	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. The following applies to BLOBs only: If IXFCSBCP is zero, the string is bit data, and should not be translated by any transformation program. The following applies to CLOBs only: If IXFCDBCP is nonzero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
BLOB_FILE, CLOB_FILE, DBCLOB_FILE	804, 808, 812	A fixed-length field containing an SQLFILE structure with the *name_length* and the *name* fields filled in. The length of the structure is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 255 bytes. The file name is in the code page indicated by IXFCSBCP. If IXFCDBCP is nonzero, the file name can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the file name is bit data and should not be translated by any transformation program. Since the length of the structure is stored in IXFCLENG, the actual length of the original LOB is lost. IXF files with columns of type BLOB_FILE, CLOB_FILE, or DBCLOB_FILE should not be used to recreate the LOB field, since the LOB will be created with a length of *sql_lobfile_len*.
CHAR	452	A fixed-length character string. The string length is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is nonzero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
DATE	384	A point in time in accordance with the Gregorian calendar. Each date is a 10-byte character string in International Standards Organization (ISO) format: *yyyy-mm-dd*. The range of the year part is 0001 to 9999. The range of the month part is 01 to 12. The range of the day part is 01 to *n*, where *n* depends on the month, using the usual rules for days of the month and leap year. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within DATE are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
DBCLOB	412	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
DECIMAL	484	A packed decimal number with precision P (as specified by the first three bytes of IXFCLENG in the column descriptor record) and scale S (as specified by the last two bytes of IXFCLENG). The length, in bytes, of a packed decimal number is (P+2)/2. The precision must be an odd number between 1 and 31, inclusive. The packed decimal number is in the internal format specified by IXFTMFRM, where packed decimal for the PC is defined to be the same as packed decimal for the System/370. IXFCSBCP and IXFCDBCP are not significant, and should be zero.
FLOATING POINT	480	Either a long (8-byte) or short (4-byte) floating point number, depending on whether IXFCLENG is set to eight or to four. The data is in the internal machine form, as specified by IXFTMFRM. IXFCSBCP and IXFCDBCP are not significant, and should be zero. Four-byte floating point is not supported by the database manager.
GRAPHIC	468	A fixed-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the number of double-byte characters in the string, and cannot exceed 127. The actual length of the string is twice the value of the IXFCLENG field, in bytes. The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
INTEGER	496	A 4-byte integer in the form specified by IXFTMFRM. It represents a whole number between -2 147 483 648 and +2 147 483 647. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
LONGVARCHAR	456	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is nonzero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
LONG VARGRAPHIC	472	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters for the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
SMALLINT	500	A 2-byte integer in the form specified by IXFTMFRM. It represents a whole number between −32 768 and +32 767. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
TIME	388	A point in time in accordance with the 24-hour clock. Each time is an 8-byte character string in ISO format: *hh.mm.ss*. The range of the hour part is 00 to 24, and the range of the other parts is 00 to 59. If the hour is 24, the other parts are 00. The smallest time is 00.00.00, and the largest is 24.00.00. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within TIME are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
TIMESTAMP	392	The date and time with microsecond precision. Each time stamp is a character string of the form *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). IXFCLENG is not used, and should contain blanks. Valid characters within TIMESTAMP are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
VARCHAR	448	A variable-length character string. The maximum length of the string, in bytes, is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string itself is preceded by a current length indicator, which is a two-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is nonzero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.

Table 11. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
VARGRAPHIC	464	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 127. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.

Not all combinations of IXFCSBCP and IXFCDBCP values for PC/IXF character or graphic columns are valid. A PC/IXF character or graphic column with an invalid (IXFCSBCP,IXFCDBCP) combination is an invalid data type.

Table 12. Valid PC/IXF Data Types

PC/IXF Data Type	Valid (IXFCSBCP,IXFCDBCP) Pairs	Invalid (IXFCSBCP,IXFCDBCP) Pairs
CHAR, VARCHAR, or LONG VARCHAR	(0,0), (x,0), or (x,y)	(0,y)
BLOB	(0,0)	(x,0), (0,y), or (x,y)
CLOB	(x,0), (x,y)	(0,0), (0,y)
GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, or DBCLOB	(0,y)	(0,0), (x,0), or (x,y)
Note: x and y are not 0.		

PC/IXF Data Type Descriptions

Table 13. Acceptable Data Type Forms for the PC/IXF File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	A BIGINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.
BLOB	A PC/IXF BLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, BLOB, or BLOB_FILE column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC BLOB column is also acceptable. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.

Table 13. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	**Form in Files Created by the Export Utility**	**Form Acceptable to the Import Utility**
CHAR	A PC/IXF CHAR column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the PC/IXF column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the length of the database column. The data is padded on the right with single-byte spaces (x'20'), if necessary. See also the “FORCEIN Option” on page 218.
CLOB	A PC/IXF CLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, CLOB, or CLOB_FILE column is acceptable if the PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.
DATE	A DATE column, identical to the database column, is created.	A PC/IXF column of type DATE is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain dates in a format consistent with the country code of the target database.

Table 13. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
DBCLOB	A PC/IXF DBCLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, DBCLOB, or DBCLOB_FILE column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.
DECIMAL	A DECIMAL column, identical to the database column, is created. The precision and scale of the column is stored in the column descriptor record.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the DECIMAL column into which they are being imported.
FLOAT	A FLOAT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are within range.
GRAPHIC (DBCS only)	A PC/IXF GRAPHIC column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the database column length. The data is padded on the right with double-byte spaces (x'8140'), if necessary. See also the “FORCEIN Option” on page 218.
INTEGER	An INTEGER column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647.

Table 13. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
LONG VARCHAR	A PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.
LONG VARGRAPHIC (DBCS only)	A PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.
SMALLINT	A SMALLINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767.

Table 13. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
TIME	A TIME column, identical to the database column, is created.	A PC/IXF column of type TIME is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain time data in a format consistent with the country code of the target database.
TIMESTAMP	A TIMESTAMP column, identical to the database column, is created.	A PC/IXF column of type TIMESTAMP is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain data in the input format for time stamps.
VARCHAR	If the maximum length of the database column is <= 254, a PC/IXF VARCHAR column is created. If the maximum length of the database column is > 254, a PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.

Table 13. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
VARGRAPHIC (DBCS only)	If the maximum length of the database column is <= 127, a PC/IXF VARGRAPHIC column is created. If the maximum length of the database column is > 127, a PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column. See also the “FORCEIN Option” on page 218.

General Rules Governing PC/IXF File Import into Databases

The database manager import utility applies the following general rules when importing a PC/IXF file in either an SBCS or a DBCS environment:

- The import utility accepts PC/IXF format files only (IXFHID = 'IXF'). IXF files of other formats cannot be imported.
- The import utility rejects a PC/IXF file with more than 1024 columns.
- The value of IXFHSBCP in the PC/IXF H record must equal the SBCS CPGID, or there must be a conversion table between the IXFHSBCP/IXFHDBCP and the SBCS/DBCS CPGID of the target database. The value of IXFHDBCP must equal either '00000', or the DBCS CPGID of the target database. If either of these conditions is not satisfied, the import utility rejects the PC/IXF file, unless the “FORCEIN Option” on page 218 is specified.
- Invalid Data Types — New Table

  Import of a PC/IXF file into a *new* table is specified by the CREATE or the REPLACE_CREATE keywords in the IMPORT command. If a PC/IXF column of an invalid data type (valid data types are defined in “PC/IXF

Data Types" on page 201) is selected for import into a new table, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.

- Invalid Data Types — Existing Table

  Import of a PC/IXF file into an *existing* table is specified by the INSERT, the INSERT_UPDATE, or the REPLACE_CREATE keywords in the IMPORT command. If a PC/IXF column of an invalid data type is selected for import into an existing table, one of two actions is possible:

  – If the target table column is nullable, all values for the invalid PC/IXF column are ignored, and the table column values are set to NULL

  – If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.
- When importing into a new table, nullable PC/IXF columns generate nullable database columns, and not nullable PC/IXF columns generate not nullable database columns.
- A not nullable PC/IXF column can be imported into a nullable database column.
- A nullable PC/IXF column can be imported into a not nullable database column. If a NULL value is encountered in the PC/IXF column, the import utility rejects the values of all columns in the PC/IXF row that contains the NULL value (the entire row is rejected), and processing continues with the next PC/IXF row. That is, no data is imported from a PC/IXF row that contains a NULL value if a target table column (for the NULL) is not nullable.
- Incompatible Columns — New Table

  If, during import to a *new* database table, a PC/IXF column is selected that is incompatible with the target database column, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.

  Note: The IMPORT "FORCEIN Option" on page 218 extends the scope of compatible columns.
- Incompatible Columns — Existing Table

  If, during import to an *existing* database table, a PC/IXF column is selected that is incompatible with the target database column, one of two actions is possible:

  – If the target table column is nullable, all values for the PC/IXF column are ignored, and the table column values are set to NULL

  – If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.

Note: The IMPORT "FORCEIN Option" on page 218 extends the scope of compatible columns.

- Invalid Values

  If, during import, a PC/IXF column value is encountered that is not valid for the target database column, the import utility rejects the values of all columns in the PC/IXF row that contains the invalid value (the entire row is rejected), and processing continues with the next PC/IXF row.
- Importing or loading PC/IXF files containing DBCS data requires that the corresponding conversion files (located in `sqllib\conv`) be installed on the client machine. The names of these conversion files contain both the source and the target code page numbers; the extension is always `.cnv`. For example, file `09320943.cnv` contains the conversion table for converting code page 932 to 943.

  If the client machine does not have the appropriate conversion files, they can be copied from a server machine to the `sqllib\conv` directory on the client machine. Be sure to copy the files from a compatible platform; for example, if the client is running on a UNIX based operating system, copy the files from a server that is also running on a UNIX based operating system.

Data Type-Specific Rules Governing PC/IXF File Import into Databases

- A valid PC/IXF numeric column can be imported into any compatible numeric database column. PC/IXF columns containing 4-byte floating point data are not imported, because this is an invalid data type.
- Database date/time columns can accept values from matching PC/IXF date/time columns (DATE, TIME, and TIMESTAMP), as well as from PC/IXF character columns (CHAR, VARCHAR, and LONG VARCHAR), subject to column length and value compatibility restrictions.
- A valid PC/IXF character column (CHAR, VARCHAR, or LONG VARCHAR) can always be imported into an *existing* database character column marked FOR BIT DATA; otherwise:
  - IXFCSBCP and the SBCS CPGID must agree
  - There must be a conversion table for the IXFCSBCP/IXFCDBCP and the SBCS/DBCS
  - One set must be all zeros (FOR BIT DATA).

  If IXFCSBCP is not zero, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the target database column.

  If either of these conditions is not satisfied, the PC/IXF and database columns are incompatible.

  When importing a valid PC/IXF character column into a *new* database table, the value of IXFCSBCP must equal either zero or the SBCS CPGID of the database, or there must be a conversion table. If IXFCSBCP is zero, IXFCDBCP must also be zero (otherwise the PC/IXF column is an invalid data type); IMPORT creates a character column marked FOR BIT DATA in the new table. If IXFCSBCP is not zero, and equals the SBCS CPGID of the database, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the database; in this case, the utility creates a character column in the new table with SBCS and DBCS CPGID values equal to those of the database. If these conditions are not satisfied, the PC/IXF and database columns are incompatible.

The “FORCEIN Option” on page 218 can be used to override code page equality checks. However, a PC/IXF character column with IXFCSBCP equal to zero and IXFCDBCP not equal to zero is an invalid data type, and cannot be imported, even if FORCEIN is specified.

- A valid PC/IXF graphic column (GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC) can always be imported into an *existing* database character column marked FOR BIT DATA, but is incompatible with all other database columns. The “FORCEIN Option” on page 218 can be used to relax this restriction. However, a PC/IXF graphic column with IXFCSBCP not equal to zero, or IXFCDBCP equal to zero, is an invalid data type, and cannot be imported, even if FORCEIN is specified.

  When importing a valid PC/IXF graphic column into a database graphic column, the value of IXFCDBCP must equal the DBCS CPGID of the target database column (that is, the double-byte code pages of the two columns must agree).
- If, during import of a PC/IXF file into an existing database table, a fixed-length string column (CHAR or GRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns are incompatible.
- If, during import of a PC/IXF file into an existing database table, a variable-length string column (VARCHAR, LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns *are* compatible. Individual values are processed according to the compatibility rules governing the database manager INSERT statement, and PC/IXF values which are too long for the target database column are invalid.
- PC/IXF values imported into a fixed-length database *character* column (that is, a CHAR column) are padded on the right with single-byte spaces (0x20), if necessary, to obtain values whose length equals that of the database column. PC/IXF values imported into a fixed-length database *graphic* column (that is, a GRAPHIC column) are padded on the right with double-byte spaces (0x8140), if necessary, to obtain values whose length equals that of the database column.

- Since PC/IXF VARCHAR columns have a maximum length of 254 bytes, a database VARCHAR column of maximum length *n*, with 254 < *n* < 4001, must be exported into a PC/IXF LONG VARCHAR column of maximum length *n*.
- Although PC/IXF LONG VARCHAR columns have a maximum length of 32 767 bytes, and database LONG VARCHAR columns have a maximum length restriction of 32 700 bytes, PC/IXF LONG VARCHAR columns of length greater than 32 700 bytes (but less than 32 768 bytes) are still valid, and can be imported into database LONG VARCHAR columns, but data may be lost.
- Since PC/IXF VARGRAPHIC columns have a maximum length of 127 bytes, a database VARGRAPHIC column of maximum length *n*, with 127 < *n* < 2001, must be exported into a PC/IXF LONG VARGRAPHIC column of maximum length *n*.
- Although PC/IXF LONG VARGRAPHIC columns have a maximum length of 16 383 bytes, and database LONG VARGRAPHIC columns have a maximum length restriction of 16 350, PC/IXF LONG VARGRAPHIC columns of length greater than 16 350 bytes (but less than 16 384 bytes) are still valid, and can be imported into database LONG VARGRAPHIC columns, but data may be lost.

Table 14 summarizes PC/IXF file import into new or existing database tables without the FORCEIN option.

Table 14. Summary of PC/IXF File Import without FORCEIN Option

PC/IXF COLUMN DATA TYPE	**DATABASE COLUMN DATA TYPE**											
	NUMERIC					**CHARACTER**			**GRAPH**	**DATETIME**		
	SMALL INT	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)[d]**	**(SBCS, DBCS)[b]**	**[b]**	**DATE**	**TIME**	**TIME STAMP**
Numeric												
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							
Character												
-(0,0)						N						
						E				E[c]	E[c]	E[c]

Table 14. Summary of PC/IXF File Import without FORCEIN Option (continued)

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[d]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP
-(SBCS,0)							N	N				
						E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)								N		E[c]	E[c]	E[c]
						E		E				
Graphic												
									N			
						E			E			
Datetime												
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Notes:

1. The table is a matrix of all valid PC/IXF and database manager data types. If a PC/IXF column can be imported into a database column, a letter is displayed in the matrix cell at the intersection of the PC/IXF data type matrix row and the database manager data type matrix column. An 'N' indicates that the utility is creating a new database table (a database column of the indicated data type is created). An 'E' indicates that the utility is importing data to an existing database table (a database column of the indicated data type is a valid target).
2. Character string data types are distinguished by code page attributes. These attributes are shown as an ordered pair (SBCS,DBCS), where:
   - SBCS is either zero or denotes a nonzero value of the single-byte code page attribute of the character data type
   - DBCS is either zero or denotes a nonzero value of the double-byte code page attribute of the character data type.
3. If the table indicates that a PC/IXF character column can be imported into a database character column, the values of their respective code page attribute pairs satisfy the rules governing code page equality.

[a] Individual values are rejected if they are out of range for the target numeric data type.

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Data type is not available in DBCS environments.

FORCEIN Option

The FORCEIN option permits import of a PC/IXF file despite code page differences between data in the PC/IXF file and the target database. It offers additional flexibility in the definition of compatible columns.

FORCEIN General Semantics

The following general semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option should be used with caution. It is usually advisable to attempt an import without this option enabled. However, because of the generic nature of the PC/IXF data interchange architecture, some PC/IXF files may contain data types or values that cannot be imported without intervention.
- Import with FORCEIN to a *new* table may yield a different result than import to an existing table. An existing table has predefined target data types for each PC/IXF data type.
- When LOB data is exported with the LOBSINFILE option, and the files move to another client with a different code page, then, unlike other data, the CLOBS and DBCLOBS in the separate files are not converted to the client code page when imported or loaded into a database.

FORCEIN Code Page Semantics

The following code page semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option disables all import utility code page comparisons.

  This rule applies to code page comparisons at the column level and at the file level as well, when importing to a new or an existing database table. At the column (for example, data type) level, this rule applies only to the following database manager and PC/IXF data types: character (CHAR, VARCHAR, and LONG VARCHAR), and graphic (GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC). The restriction follows from the fact that code page attributes of other data types are not relevant to the interpretation of data type values.
- The FORCEIN option does not disable inspection of code page attributes to determine data types.

  For example, the database manager allows a CHAR column to be declared with the FOR BIT DATA attribute. Such a declaration sets both the SBCS CPGID and the DBCS CPGID of the column to zero; it is the zero value of these CPGIDs that identifies the column values as bit strings (rather than character strings).
- The FORCEIN option does not imply code page translation.

  Values of data types that are sensitive to the FORCEIN option are copied "as is". No code point mappings are employed to account for a change of code page environments. Padding of the imported value with spaces may be necessary in the case of fixed length target columns.
- When data is imported to an *existing* table using the FORCEIN option:

  - The code page value of the target database table and columns always prevails.
  - The code page value of the PC/IXF file and columns is ignored.

  This rule applies whether or not the FORCEIN option is used. The database manager does not permit changes to a database or a column code page value once a database is created.

- When importing to a *new* table using the FORCEIN option:
  - The code page value of the target database prevails.
  - PC/IXF character columns with IXFCSBCP = IXFCDBCP = 0 generate table columns marked FOR BIT DATA.
  - All other PC/IXF character columns generate table character columns with SBCS and DBCS CPGID values equal to those of the database.
  - PC/IXF graphic columns generate table graphic columns with an SBCS CPGID of "undefined", and a DBCS CPGID equal to that of the database (DBCS environment only).

FORCEIN Example

Consider a PC/IXF CHAR column with IXFCSBCP = '00897' and IXFCDBCP = '00301'. This column is to be imported into a database CHAR column whose SBCS CPGID = '00850' and DBCS CPGID = '00000'. Without FORCEIN, the utility terminates, and no data is imported, or the PC/IXF column values are ignored, and the database column contains NULLs (if the database column is nullable). With FORCEIN, the utility proceeds, ignoring code page incompatibilities. If there are no other data type incompatibilities (such as length, for example), the values of the PC/IXF column are imported "as is", and become available for interpretation under the database column code page environment.

The following table shows:

- The code page attributes of a column created in a *new* database table when a PC/IXF file data type with specified code page attributes is imported
- That the import utility rejects PC/IXF data types if they invalid or incompatible.

Table 15. Summary of Import Utility Code Page Semantics (New Table). This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without the FORCEIN option.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without FORCEIN	With FORCEIN
SBCS		
(0,0)	(0,0)	(0,0)
(a,0)	(a,0)	(a,0)
(x,0)	reject	(a,0)
(x,y)	reject	(a,0)
(a,y)	reject	(a,0)
(0,y)	reject	(0,0)
DBCS		
(0,0)	(0,0)	(0,0)
(a,0)	(a,b)	(a,b)
(x,0)	reject	(a,b)
(a,b)	(a,b)	(a,b)
(x,y)	reject	(a,b)
(a,y)	reject	(a,b)
(x,b)	reject	(a,b)
(0,b)	(-,b)	(-,b)
(0,y)	reject	(-,b)

Table 15. Summary of Import Utility Code Page Semantics (New Table) (continued). This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without the FORCEIN option.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without FORCEIN	With FORCEIN

Notes:

1. Code page attributes of a PC/IXF data type are shown as an ordered pair, where x represents a nonzero single-byte code page value, and y represents a nonzero double-byte code page value. A '-' represents an undefined code page value.
2. The use of different letters in various code page attribute pairs is deliberate. Different letters imply different values. For example, if a PC/IXF data type is shown as (x,y), and the database column as (a,y), x does not equal a, but the PC/IXF file and the database have the same double-byte code page value y.
3. Only character and graphic data types are affected by the FORCEIN code page semantics.
4. It is assumed that the database containing the new table has code page attributes of (a,0); therefore, all character columns in the new table must have code page attributes of either (0,0) or (a,0).

   In a DBCS environment, it is assumed that the database containing the new table has code page attributes of (a,b); therefore, all graphic columns in the new table must have code page attributes of (-,b), and all character columns must have code page attributes of (a,b). The SBCS CPGID is shown as '-', because it is undefined for graphic data types.
5. The data type of the result is determined by the rules described in “FORCEIN Data Type Semantics” on page 224.
6. The `reject` result is a reflection of the rules for invalid or incompatible data types (see “General Rules Governing PC/IXF File Import into Databases” on page 213).

The following table shows:

- That the import utility accepts PC/IXF data types with various code page attributes into an *existing* table column (the *target* column) having the specified code page attributes
- That the import utility does not permit a PC/IXF data type with certain code page attributes to be imported into an *existing* table column having the code page attributes shown. The utility rejects PC/IXF data types if they are invalid or incompatible.

Table 16. Summary of Import Utility Code Page Semantics (Existing Table). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	With FORCEIN
SBCS			
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept
(0,0)	(a,0)	null or reject	accept
(a,0)	(a,0)	accept	accept
(x,0)	(a,0)	null or reject	accept
(x,y)	(a,0)	null or reject	accept
(a,y)	(a,0)	null or reject	accept
(0,y)	(a,0)	null or reject	null or reject
DBCS			
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(a,b)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(x,b)	(0,0)	accept	accept
(0,b)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept
(0,0)	(a,b)	null or reject	accept
(a,0)	(a,b)	accept	accept
(x,0)	(a,b)	null or reject	accept
(a,b)	(a,b)	accept	accept

Table 16. Summary of Import Utility Code Page Semantics (Existing Table) (continued). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	With FORCEIN
(x,y)	(a,b)	null or reject	accept
(a,y)	(a,b)	null or reject	accept
(x,b)	(a,b)	null or reject	accept
(0,b)	(a,b)	null or reject	null or reject
(0,y)	(a,b)	null or reject	null or reject
(0,0)	(-,b)	null or reject	accept
(a,0)	(-,b)	null or reject	null or reject
(x,0)	(-,b)	null or reject	null or reject
(a,b)	(-,b)	null or reject	null or reject
(x,y)	(-,b)	null or reject	null or reject
(a,y)	(-,b)	null or reject	null or reject
(x,b)	(-,b)	null or reject	null or reject
(0,b)	(-,b)	accept	accept
(0,y)	(-,b)	null or reject	accept

Notes:

1. See the notes for Table 15 on page 221.
2. The null or reject result is a reflection of the rules for invalid or incompatible data types (see “General Rules Governing PC/IXF File Import into Databases” on page 213).

FORCEIN Data Type Semantics

The FORCEIN option permits import of certain PC/IXF columns into target database columns of unequal and otherwise incompatible data types. The following data type semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment (except where noted):

- In SBCS environments, the FORCEIN option permits import of:
  - A PC/IXF BIT data type (IXFCSBCP = 0 = IXFCDBCP for a PC/IXF character column) into a database character column (nonzero SBCS CPGID, and DBCS CPGID = 0); existing tables only

  - A PC/IXF MIXED data type (nonzero IXFCSBCP and IXFCDBCP) into a database character column; both new and existing tables
  - A PC/IXF GRAPHIC data type into a database FOR BIT DATA column (SBCS CPGID = 0 = DBCS CPGID); new tables only (this is always permitted for existing tables).
- The FORCEIN option does not extend the scope of valid PC/IXF data types.

  PC/IXF columns with data types not defined as valid in “PC/IXF Data Types” on page 201 are invalid for import with or without the FORCEIN option.
- In DBCS environments, the FORCEIN option permits import of:
  - A PC/IXF BIT data type into a database character column
  - A PC/IXF BIT data type into a database graphic column; however, if the PC/IXF BIT column is of fixed length, that length must be even. A fixed length PC/IXF BIT column of odd length is not compatible with a database graphic column. A varying-length PC/IXF BIT column *is* compatible whether its length is odd or even, although an odd-length value from a varying-length column is an invalid value for import into a database graphic column
  - A PC/IXF MIXED data type into a database character column.

Table 17 summarizes PC/IXF file import into new or existing database tables with the FORCEIN option.

Table 17. Summary of PC/IXF File Import with FORCEIN Option

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[e]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP
Numeric												
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							

Table 17. Summary of PC/IXF File Import with FORCEIN Option (continued)

PC/IXF COLUMN DATA TYPE	DATABASE COLUMN DATA TYPE											
	NUMERIC					CHARACTER			GRAPH	DATETIME		
	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[e]	(SBCS, DBCS)[b]	[b]	DATE	TIME	TIME STAMP
Character												
-(0,0)						N						
						E	E w/F	E w/F	E w/F	E[c]	E[c]	E[c]
-(SBCS,0)							N	N				
						E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)							N w/F[d]	N		E[c]	E[c]	E[c]
						E	E w/F	E				
Graphic												
						N w/F[d]			N			
						E			E			
Datetime												
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Note: If a PC/IXF column can be imported into a database column only with the FORCEIN option, the string 'w/F' is displayed together with an 'N' or an 'E'. An 'N' indicates that the utility is creating a new database table; an 'E' indicates that the utility is importing data to an existing database table. The FORCEIN option affects compatibility of character and graphic data types only.

[a] Individual values are rejected if they are out of range for the target numeric data type.

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Applies only if the source PC/IXF data type is not supported by the target database.

[e] Data type is not available in DBCS environments.

Differences between Version 1 PC/IXF and Version 0 System/370 IXF

The following describes differences between Version 1 PC/IXF, used by the database manager, and Version 0 System/370 IXF, used by several host database products:

- PC/IXF files are ASCII, rather than EBCDIC oriented. PC/IXF files have significantly expanded code page identification, including new code page identifiers in the H record, and the use of actual code page values in the column descriptor records. There is also a mechanism for marking columns of character data as FOR BIT DATA. FOR BIT DATA columns are of special significance, because transforms which convert a PC/IXF file format to or from any other IXF or database file format cannot perform any code page translation on the values contained in FOR BIT DATA columns.
- Only the machine data form is permitted; that is, the IXFTFORM field must always contain the value M. Furthermore, the machine data must be in PC forms; that is, the IXFTMFRM field must contain the value PC. This means that integers, floating point numbers, and decimal numbers in data portions of PC/IXF data records must be in PC forms.
- Application (A) records are permitted anywhere after the H record in a PC/IXF file. They are not counted when the value of the IXFHHCNT field is computed.
- Every PC/IXF record begins with a record length indicator. This is a 6-byte character representation of an integer value containing the length, in bytes, of the PC/IXF record not including the record length indicator itself; that is, the total record length minus 6 bytes. The purpose of the record length field is to enable PC programs to identify record boundaries.
- To facilitate the compact storage of variable-length data, and to avoid complex processing when a field is split into multiple records, PC/IXF does not support Version 0 IXF X records, but does support D record identifiers. Whenever a variable-length field or a nullable field is the last field in a data D record, it is not necessary to write the entire maximum length of the field to the PC/IXF file.

Worksheet File Format (WSF)

Lotus 1-2-3 and Symphony products use the same basic format, with additional functions added at each new release. The database manager supports the subset of the worksheet records that are the same for all the Lotus products. That is, for the releases of Lotus 1-2-3 and Symphony products supported by the database manager, all file names with any three-character extension are accepted; for example: WKS, WK1, WRK, WR1, WJ2.

Worksheet File Format (WSF)

Each WSF file represents one worksheet. The database manager uses the following conventions to interpret worksheets and to provide consistency in worksheets generated by its export operations:

- Cells in the first row (ROW value 0) are reserved for descriptive information about the entire worksheet. All data within this row is optional. It is ignored during import.
- Cells in the second row (ROW value 1) are used for column labels.
- The remaining rows are data rows (records, or rows of data from the table).
- Cell values under any column heading are values for that particular column or field.
- A NULL value is indicated by the absence of a real cell content record (for example, no integer, number, label, or formula record) for a particular column within a row of cell content records.

  Note: A row of NULLs will be neither imported nor exported.

To create a file that is compliant with the WSF format during an export operation, some loss of data may occur.

WSF files use a Lotus code point mapping that is not necessarily the same as existing code pages supported by DB2. As a result, when importing or exporting a WSF file, data is converted from the Lotus code points to or from the code points used by the application code page. DB2 supports conversion between the Lotus code points and code points defined by code pages 437, 819, 850, 860, 863, and 865.

Note: For multi-byte character set users, no conversions are performed.

Appendix D. Warning, Error, and Completion Messages

Messages generated by the various data movement utilities are included among the SQL messages. These messages are generated by the database manager when a warning or error condition has been detected. Each message has a message identifier that consists of a prefix (SQL) and a four- or five-digit message number. There are three message types: notification, warning, and critical. Message identifiers ending with an `N` are error messages. Those ending with a `W` indicate warning or informational messages. Message identifiers ending with a `C` indicate critical system errors.

The message number is also referred to as the *SQLCODE*. The SQLCODE is passed to the application as a positive or negative number, depending on its message type (N, W, or C). N and C yield negative values, whereas W yields a positive value. DB2 returns the SQLCODE to the application, and the application can get the message associated with the SQLCODE. DB2 also returns an *SQLSTATE* value for conditions that could be the result of an SQL statement. Some SQLCODE values have associated SQLSTATE values.

For detailed information about all of the DB2 messages, see the *Message Reference*. You can use the information contained in this book to identify an error or problem, and to resolve the problem by using the appropriate recovery action. This information can also be used to understand where messages are generated and logged.

SQL messages, and the message text associated with SQLSTATE values, are also accessible from the operating system command line. To access help for these error messages, enter the following at the operating system command prompt:

```
db2 ? SQLnnnnn
```

where *nnnnn* represents the message number.

The message identifier accepted as a parameter for the **db2** command is not case sensitive, and the terminating letter is not required. Therefore, the following commands will produce the same result:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

If the message text is too long for your screen, use the following command (on UNIX based operating systems and others that support the ″more″ pipe):

```
db2 ? SQLnnnnn | more
```

You can also redirect the output to a file which can then be browsed.

Help can also be invoked from interactive input mode. To access this mode, enter the following at the operating system command prompt:

```
db2
```

To get DB2 message help in this mode, type the following at the command prompt (`db2 =>`):

```
? SQLnnnnn
```

The message text associated with SQLSTATEs can be retrieved by issuing:

```
db2 ? nnnnn
 or
db2 ? nn
```

where *nnnnn* is a five-character SQLSTATE value (alphanumeric), and *nn* is a two-digit SQLSTATE class code (the first two digits of the SQLSTATE value).

Appendix E. How the DB2 Library Is Structured

The DB2 Universal Database library consists of SmartGuides, online help, books and sample programs in HTML format. This section describes the information that is provided, and how to access it.

To access product information online, you can use the Information Center. You can view task information, DB2 books, troubleshooting information, sample programs, and DB2 information on the Web. See "Accessing Information with the Information Center" on page 242 for details.

Completing Tasks with SmartGuides

SmartGuides help you complete some administration tasks by taking you through each task one step at a time. SmartGuides are available through the Control Center and the Client Configuration Assistant. The following table lists the SmartGuides.

Note: Create Database, Index, and Configure Multisite Update SmartGuide are available for the partitioned database environment.

SmartGuide	Helps You to...	How to Access...
Add Database	Catalog a database on a client workstation.	From the Client Configuration Assistant, click **Add**.
Back up Database	Determine, create, and schedule a backup plan.	From the Control Center, click with the right mouse button on the database you want to back up and select **Backup**->**Database using SmartGuide**.
Configure Multisite Update SmartGuide	Perform a multi-site update, a distributed transaction, or a two-phase commit.	From the Control Center, click with the right mouse button on the **Database** icon and select **Multisite Update**.
Create Database	Create a database, and perform some basic configuration tasks.	From the Control Center, click with the right mouse button on the **Databases** icon and select **Create**->**Database using SmartGuide**.

SmartGuide	Helps You to...	How to Access...
Create Table	Select basic data types, and create a primary key for the table.	From the Control Center, click with the right mouse button on the **Tables** icon and select **Create**->**Table using SmartGuide**.
Create Table Space	Create a new table space.	From the Control Center, click with the right mouse button on the **Table spaces** icon and select **Create**->**Table space using SmartGuide**.
Index	Advise which indexes to create and drop for all your queries.	From the Control Center, click with the right mouse button on the **Index** icon and select **Create**->**Index using SmartGuide**.
Performance Configuration	Tune the performance of a database by updating configuration parameters to match your business requirements.	From the Control Center, click with the right mouse button on the database you want to tune and select **Configure using SmartGuide**.
Restore Database	Recover a database after a failure. It helps you understand which backup to use, and which logs to replay.	From the Control Center, click with the right mouse button on the database you want to restore and select **Restore**->**Database using SmartGuide**.

Accessing Online Help

Online help is available with all DB2 components. The following table describes the various types of help. You can also access DB2 information through the Information Center. For information see “Accessing Information with the Information Center” on page 242.

Type of Help	Contents	How to Access...
Command Help	Explains the syntax of commands in the command line processor.	From the command line processor in interactive mode, enter: ? *command* where *command* is a keyword or the entire command. For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.

Type of Help	Contents	How to Access...
Control Center Help *Client Configuration Assistant Help* *Event Analyzer Help* *Command Center Help*	Explains the tasks you can perform in a window or notebook. The help includes prerequisite information you need to know, and describes how to use the window or notebook controls.	From a window or notebook, click the **Help** push button or press the F1 key.
Message Help	Describes the cause of a message, and any action you should take.	From the command line processor in interactive mode, enter: `?` *XXXnnnnn* where *XXXnnnnn* is a valid message identifier. For example, `? SQL30081` displays help about the SQL30081 message. To view message help one screen at a time, enter: `?` *XXXnnnnn* `| more` To save message help in a file, enter: `?` *XXXnnnnn* `>` *filename.ext* where *filename.ext* is the file where you want to save the message help.
SQL Help	Explains the syntax of SQL statements.	From the command line processor in interactive mode, enter: `help` *statement* where *statement* is an SQL statement. For example, **help** SELECT displays help about the SELECT statement. **Note:** SQL help is not available on UNIX-based platforms.
SQLSTATE Help	Explains SQL states and class codes.	From the command line processor in interactive mode, enter: **?** *sqlstate* or **?** *class-code* where *sqlstate* is a valid five-digit SQL state and *class-code* is the first two digits of the SQL state. For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.

DB2 Information – Hardcopy and Online

The table in this section lists the DB2 books. They are divided into two groups:

Cross-platform books
These books contain the common DB2 information for all platforms.

Platform-specific books
These books are for DB2 on a specific platform. For example, there are separate *Quick Beginnings* books for DB2 on OS/2, on Windows NT, and on the UNIX-based platforms.

Cross-platform sample programs in HTML
These samples are the HTML version of the sample programs that are installed with the SDK. They are for informational purposes and do not replace the actual programs.

Most books are available in HTML and PostScript format, or you can choose to order a hardcopy from IBM. The exceptions are noted in the table.

On OS/2 and Windows platforms, HTML documentation files can be installed under the doc\html subdirectory. Depending on the language of your system, some files may be in that language, and the remainder are in English.

On UNIX platforms, you can install multiple language versions of the HTML documentation files under the doc/%L/html subdirectories. Any documentation that is not available in a national language is shown in English.

You can obtain DB2 books and access information in a variety of different ways:

View	See “Viewing Online Information” on page 241.
Search	See “Searching Online Information” on page 244.
Print	See “Printing the PostScript Books” on page 244.
Order	See “Ordering the Printed Books” on page 245.

Name	Description	Form Number File Name for Online Book	HTML Directory
	Cross-Platform Books		

Name	Description	Form Number File Name for Online Book	HTML Directory
Administration Guide	*Administration Guide, Design and Implementation* contains information required to design, implement, and maintain a database. It also describes database access using the Control Center(whether local or in a client/server environment), auditing, database recovery, distributed database support, and high availability. *Administration Guide, Performance* contains information that focuses on the database environment, such as application performance evaluation and tuning. You can order both volumes of the *Administration Guide* in the English language in North America using the form number SBOF-8922.	Volume 1 SC09-2839 db2d1x60 Volume 2 SC09-2840 db2d2x60	db2d0
Administrative API Reference	Describes the DB2 application programming interfaces (APIs) and data structures you can use to manage your databases. Explains how to call APIs from your applications.	SC09-2841 db2b0x60	db2b0
Application Building Guide	Provides environment setup information and step-by-step instructions about how to compile, link, and run DB2 applications on Windows, OS/2, and UNIX-based platforms. This book combines the *Building Applications* books for the OS/2, Windows, and UNIX-based environments.	SC09-2842 db2axx60	db2ax
APPC, CPI-C and SNA Sense Codes	Provides general information about APPC, CPI-C, and SNA sense codes that you may encounter when using DB2 Universal Database products. **Note:** Available in HTML format only.	No form number db2apx60	db2ap

Name	Description	Form Number File Name for Online Book	HTML Directory
Application Development Guide	Explains how to develop applications that access DB2 databases using embedded SQL or JDBC, how to write stored procedures, user-defined types, user-defined functions, and how to use triggers. It also discusses programming techniques and performance considerations. This book was formerly known as the *Embedded SQL Programming Guide*.	SC09-2845 db2a0x60	db2a0
CLI Guide and Reference	Explains how to develop applications that access DB2 databases using the DB2 Call Level Interface, a callable SQL interface that is compatible with the Microsoft ODBC specification.	SC09-2843 db2l0x60	db2l0
Command Reference	Explains how to use the command line processor, and describes the DB2 commands you can use to manage your database.	SC09-2844 db2n0x60	db2n0
Data Movement Utilities Guide and Reference	Explains how to use the Load, Import, Export, Autoloader, and Data Propogation utilities to work with the data in the database.	SC09-2858 db2dmx60	db2dm
DB2 Connect Personal Edition Quick Beginnings	Provides planning, installing, and configuring information for DB2 Connect Personal Edition.	GC09-2830 db2c1x60	db2c1
DB2 Connect User's Guide	Provides concepts, programming and general usage information about the DB2 Connect products.	SC09-2838 db2c0x60	db2c0
Connectivity Supplement	Provides setup and reference information on how to use DB2 for AS/400, DB2 for OS/390, DB2 for MVS, or DB2 for VM as DRDA application requesters with DB2 Universal Database servers, and on how to use DRDA application servers with DB2 Connect application requesters. **Note:** Available in HTML and PostScript formats only.	No form number db2h1x60	db2h1
Glossary	Provides a comprehensive list of all DB2 terms and definitions. **Note:** Available in HTML format only.	No form number db2t0x50	db2t0

Name	Description	Form Number File Name for Online Book	HTML Directory
Installation and Configuration Supplement	Guides you through the planning, installation, and set up of platform-specific DB2 clients. This supplement contains information on binding, setting up client and server communications, DB2 GUI tools, DRDA AS, distributed installation, and the configuration of distributed requests and access methods to heterogeneous data sources.	GC09-2857 db2iyx60	db2iy
Message Reference	Lists messages and codes issued by DB2, and describes the actions you should take.	GC09-2846 db2m0x60	db2m0
Replication Guide and Reference	Provides planning, configuration, administration, and usage information for the IBM Replication tools supplied with DB2.	SC26-9642 db2e0x60	db2e0
SQL Getting Started	Introduces SQL concepts, and provides examples for many constructs and tasks.	SC09-2856 db2y0x60	db2y0
SQL Reference, Volume 1 and Volume 2	Describes SQL syntax, semantics, and the rules of the language. Also includes information about release-to-release incompatibilities, product limits, and catalog views. You can order both volumes of the *SQL Reference* in the English language in North America with the form number SBOF-8923.	SBOF-8923 Volume 1 db2s1x60 Volume 2 db2s2x60	db2s0
System Monitor Guide and Reference	Describes how to collect different kinds of information about databases and the database manager. Explains how to use the information to understand database activity, improve performance, and determine the cause of problems.	SC09-2849 db2f0x60	db2f0
Troubleshooting Guide	Helps you determine the source of errors, recover from problems, and use diagnostic tools in consultation with DB2 Customer Service.	S10J-8169	db2p0

Name	Description	Form Number File Name for Online Book	HTML Directory
What's New	Describes the new features, functions, and enhancements in DB2 Universal Database, Version 6.0, including information about Java-based tools.	SC09-2851 db2q0x60	db2q0
	Platform-Specific Books		
Administering Satellites Guide and Reference	Provides planning, configuration, administration, and usage information for satellites.	GC09-2821 db2dsx60	db2ds
DB2 Personal Edition Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database Personal Edition on the OS/2, Windows 95, and Windows NT operating systems.	GC09-2831 db2i1x60	db2i1
DB2 for OS/2 Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the OS/2 operating system. Also contains installing and setup information for many supported clients.	GC09-2834 db2i2x60	db2i2
DB2 for UNIX Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on UNIX-based platforms. Also contains installing and setup information for many supported clients.	GC09-2836 db2ixx60	db2ix
DB2 for Windows NT Quick Beginnings	Provides planning, installation, migration, and configuration information for DB2 Universal Database on the Windows NT operating system. Also contains installing and setup information for many supported clients.	GC09-2835 db2i6x60	db2i6
DB2 Enterprise - Extended Edition for UNIX Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for UNIX. Also contains installing and setup information for many supported clients.	GC09-2832 db2v3x60	db2v3

Name	Description	Form Number File Name for Online Book	HTML Directory
DB2 Enterprise - Extended Edition for Windows NT Quick Beginnings	Provides planning, installation, and configuration information for DB2 Enterprise - Extended Edition for Windows NT. Also contains installing and setup information for many supported clients.	GC09-2833 db2v6x60	db2v6
DB2 Connect Enterprise Edition for OS/2 and Windows NT Quick Beginnings	Provides planning, migration, installation, and configuration information for DB2 Connect Enterprise Edition on the OS/2 and Windows NT operating systems. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings*.	GC09-2828 db2c6x60	db2c6
DB2 Connect Enterprise Edition for UNIX Quick Beginnings	Provides planning, migration, installation, configuration, and usage information for DB2 Connect Enterprise Edition in UNIX-based platforms. Also contains installation and setup information for many supported clients. This book was formerly part of the *DB2 Connect Enterprise Edition Quick Beginnings*.	GC09-2829 db2cyx60	db2cy
DB2 Data Links Manager for AIX Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for AIX.	GC09-2837 db2z0x60	db2z0
DB2 Data Links Manager for Windows NT Quick Beginnings	Provides planning, installation, configuration, and task information for DB2 Data Links Manager for Windows NT.	GC09-2827 db2z6x60	db2z6
DB2 Query Patroller Administration Guide	Provides administration information on DB2 Query Patrol.	SC09-2859 db2dwx60	db2dw
DB2 Query Patroller Installation Guide	Provides installation information on DB2 Query Patrol.	GC09-2860 db2iwx60	db2iw
DB2 Query Patroller User's Guide	Describes how to use the tools and functions of the DB2 Query Patrol.	SC09-2861 db2wwx60	db2ww

Name	Description	Form Number File Name for Online Book	HTML Directory
	Cross-Platform Sample Programs in HTML		
Sample programs in HTML	Provides the sample programs in HTML format for the programming languages on all platforms supported by DB2 for informational purposes (not all samples are available in all languages). Only available when the SDK is installed. See *Application Building Guide* for more information on the actual programs. **Note:** Available in HTML format only.	No form number	db2hs/c db2hs/cli db2hs/clp db2hs/cpp db2hs/cobol db2hs/cobol_mf db2hs/fortran db2hs/java db2hs/rexx

Notes:

1. The character in the sixth position of the file name indicates the language of a book. For example, the file name `db2d0e60` indicates that the *Administration Guide* is in English. The following letters are used in the file names to indicate the language of a book:

Language	Identifier
Brazilian Portuguese	b
Bulgarian	u
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Russian	r
Simp. Chinese	c
Slovenian	l
Spanish	z

Swedish	s
Trad. Chinese	t
Turkish	m

2. For late breaking information that could not be included in the DB2 books:
   - On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/%L directory, where `%L` is the locale name and DB2DIR is:
     - /usr/lpp/db2_06_01 on AIX
     - /opt/IBMdb2/V6.1 on HP-UX, Solaris, SCO UnixWare 7, and Silicon Graphics IRIX
     - /usr/IBMdb2/V6.1 on Linux.
   - On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.
   - Under Windows `Start` menu

Viewing Online Information

The manuals included with this product are in Hypertext Markup Language (HTML) softcopy format. Softcopy format enables you to search or browse the information, and provides hypertext links to related information. It also makes it easier to share the library across your site.

You can view the online books or sample programs with any browser that conforms to HTML Version 3.2 specifications.

To view online books or sample programs on all platforms other than SCO UnixWare 7:

- If you are running DB2 administration tools, use the Information Center. See "Accessing Information with the Information Center" on page 242 for details.
- Select the `Open Page` menu item of your Web browser. The page you open contains descriptions of and links to DB2 information:
  - On UNIX-based platforms, open the following page:

    ```
    file:/INSTHOME/sqllib/doc/%L/html/index.htm
    ```

    where *%L* is the locale name.
  - On other platforms, open the following page:

    ```
    sqllib\doc\html\index.htm
    ```

    The path is located on the drive where DB2 is installed.

If you have not installed the Information Center, you can open the page by double-clicking on the **DB2 Online Books** icon. Depending on the system you are using, the icon is in the main product folder or the Windows Start menu.

To view online books or sample programs on the SCO UnixWare 7:

- DB2 Universal Database for SCO UnixWare 7 uses the native SCOhelp utility to search the DB2 information. You can access SCOhelp by the following methods:
  - entering the ″scohelp″ command on the command line,
  - selecting the `Help` menu in the Control Panel of the CDE desktop or
  - selecting `Help` in the Root menu of the Panorama desktop

  For more information on SCOhelp, refer to the *Installation and Configuration Supplement*.

Accessing Information with the Information Center

The Information Center provides quick access to DB2 product information. The Information Center is available on all platforms on which the DB2 administration tools are available.

Depending on your system, you can access the Information Center from the:

- Main product folder
- Toolbar in the Control Center
- Windows Start menu
- Help menu of the Control Center

The Information Center provides the following kinds of information. Click the appropriate tab to look at the information:

Tasks	Lists tasks you can perform using DB2.
Reference	Lists DB2 reference information, such as keywords, commands, and APIs.
Books	Lists DB2 books.
Troubleshooting	Lists categories of error messages and their recovery actions.
Sample Programs	Lists sample programs that come with the DB2 Software Developer’s Kit. If the Software Developer’s Kit is not installed, this tab is not displayed.
Web	Lists DB2 information on the World Wide

Web. To access this information, you must have a connection to the Web from your system.

When you select an item in one of the lists, the Information Center launches a viewer to display the information. The viewer might be the system help viewer, an editor, or a Web browser, depending on the kind of information you select.

The Information Center provides some search capabilities, so you can look for specific topics, and filter capabilities to limit the scope of your searches.

For a full text search, click the Search button of the Information Center follow the *Search DB2 Books* link in each HTML file.

The HTML search server is usually started automatically. If a search in the HTML information does not work, you may have to start the search server by double-clicking its icon on the Windows or OS/2 desktop.

Refer to the release notes if you experience any other problems when searching the HTML information.

Note: Search function is not available in the Linux and Silicon Graphics environments.

Setting Up a Document Server

By default, the DB2 information is installed on your local system. This means that each person who needs access to the DB2 information must install the same files. To have the DB2 information stored in a single location, use the following instructions:

1. Copy all files and subdirectories from \sqllib\doc\html on your local system to a Web server. Each book has its own subdirectory containing all the necessary HTML and GIF files that make up the book. Ensure that the directory structure remains the same.
2. Configure the Web server to look for the files in the new location. For information, see the NetQuestion Appendix in *Installation and Configuration Supplement*.
3. If you are using the Java version of the Information Center, you can specify a base URL for all HTML files. You should use the URL for the list of books.
4. Once you are able to view the book files, you should bookmark commonly viewed topics. Among those, you will probably want to bookmark the following pages:

- List of books
- Tables of contents of frequently used books
- Frequently referenced articles, such as the *ALTER TABLE* topic
- The Search form

For information about setting up a search, see the NetQuestion Appendix in *Installation and Configuration Supplement* book.

Searching Online Information

To search for information in the HTML books, you can do the following:

- Click on **Search the DB2 Books** at the bottom of any page in the HTML books. Use the search form to find a specific topic. This function is not available in the Linux or Silicon Graphics IRIX environments.
- Click on **Index** at the bottom of any page in an HTML book. Use the index to find a specific topic in the book.
- Display the table of contents or index of the HTML book, and then use the find function of the Web browser to find a specific topic in the book.
- Use the bookmark function of the Web browser to quickly return to a specific topic.
- Use the search function of the Information Center to find specific topics. See “Accessing Information with the Information Center” on page 242 for details.

Printing the PostScript Books

If you prefer to have printed copies of the manuals, you can decompress and print PostScript versions. For the file name of each book in the library, see the table in “DB2 Information – Hardcopy and Online” on page 234. Specify the full path name for the file you intend to print.

On OS/2 and Windows platforms:

1. Copy the compressed PostScript files to a hard drive on your system. The files have a file extension of .exe and are located in the x:\doc\language\books\ps directory, where x: is the letter representing the CD-ROM drive and *language* is the two-character country code that represents your language (for example, EN for English).
2. Decompress the file that corresponds to the book that you want. Each compressed book is a self-extracting executable file. To decompress the

book, simply run it as you would run any other executable program. The result from this step is a printable PostScript file with a file extension of .ps.

3. Ensure that your default printer is a PostScript printer capable of printing Level 1 (or equivalent) files.
4. Enter the following command from a command line:

   ```
   print filename.ps
   ```

On UNIX-based platforms:

1. Mount the CD-ROM. Refer to your *Quick Beginnings* manual for the procedures to mount the CD-ROM.
2. Change to /cdrom/doc/%L/ps directory on the CD-ROM, where */cdrom* is the mount point of the CD-ROM and *%L* is the name of the desired locale. The manuals will be installed in the previously-mentioned directory with file names ending with .ps.Z.
3. Decompress and print the manual you require using the following command:
   - For AIX:

     ```
     zcat filename | qprt -P PSPrinter_queue
     ```
   - For HP-UX, Solaris, or SCO UnixWare 7:

     ```
     zcat filename | lp -d PSPrinter_queue
     ```
   - For Linux:

     ```
     zcat filename | lpr -P PSPrinter_queue
     ```
   - For Silicon Graphics IRIX:

     ```
     zcat < filename | lp -d PSPrinter_queue
     ```

   where *filename* is the full path name and extension of the compressed PostScript file and *PSprinter_queue* is the name of the PostScript printer queue.

   For example, to print the English version of *DB2 for UNIX Quick Beginnings* on AIX, you can use the following command:

   ```
   zcat /cdrom/doc/en/ps/db2ixe60.ps.Z || qprt -P ps1
   ```

Ordering the Printed Books

You can order the printed DB2 manuals either as a set or individually. There are three sets of books available. The form number for the entire set of DB2 books is SBOF-8926-00. The form number for the books listed under the heading ″Cross-Platform Books″ is SBOF-8924-00.

Note: These form numbers only apply if you are ordering books that are printed in the English language in North America.

You can also order books individually by the form number listed in “DB2 Information – Hardcopy and Online” on page 234. To order printed versions, contact your IBM authorized dealer or marketing representative, or phone 1-800-879-2755 in the United States or 1-800-IBM-4YOU in Canada.

Appendix F. Notices

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent product, program or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the

IBM Director of Licensing
IBM Corporation, North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

This publication may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States and/or other countries:

ACF/VTAM
ADSTAR
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
Extended Services
FFST
First Failure Support Technology
IBM
IMS
LAN Distance
MVS/ESA
MVS/XA
OS/400
OS/390
OS/2
PowerPC
QMF
RACF
RISC System/6000
SP
SQL/DS
SQL/400
S/370
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WIN-OS/2

Trademarks of Other Companies

The following terms are trademarks or registered trademarks of the companies listed:

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

HP-UX is a trademark of Hewlett-Packard.

Java, HotJava, Solaris, Solstice, and Sun are trademarks of Sun Microsystems, Inc.

Microsoft, Windows, Windows NT, Visual Basic, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium, and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark in the United States, other countries or both and is licensed exclusively through X/Open Company Limited.

Other company, product, or service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Index

Contacting IBM

This section lists ways you can get more information from IBM.

If you have a technical problem, please take the time to review and carry out the actions suggested by the *Troubleshooting Guide* before contacting DB2 Customer Support. Depending on the nature of your problem or concern, this guide will suggest information you can gather to help us to serve you better.

For information or to order any of the DB2 Universal Database products contact an IBM representative at a local branch office or contact any authorized IBM software remarketer.

Telephone

If you live in the U.S.A., call one of the following numbers:

- 1-800-237-5511 to learn about available service options.
- 1-800-IBM-CALL (1-800-426-2255) or 1-800-3IBM-OS2 (1-800-342-6672) to order products or get general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, see Appendix A of the IBM Software Support Handbook. You can access this document by accessing the following page:

```
http://www.ibm.com/support/
```

then performing a search using the keyword “handbook”.

Note that in some countries, IBM-authorized dealers should contact their dealer support structure instead of the IBM Support Center.

World Wide Web

http://www.software.ibm.com/data/
http://www.software.ibm.com/data/db2/library/

The DB2 World Wide Web pages provide current DB2 information about news, product descriptions, education schedules, and more. The DB2 Product and Service Technical Library provides access to frequently asked questions, fixes, books, and up-to-date DB2 technical information. (Note that this information may be in English only.)

Anonymous FTP Sites

ftp.software.ibm.com

Log on as anonymous. In the directory /ps/products/db2, you can find demos, fixes, information, and tools concerning DB2 and many related products.

Internet Newsgroups

comp.databases.ibm-db2, bit.listserv.db2-l

These newsgroups are available for users to discuss their experiences with DB2 products.

CompuServe

GO IBMDB2 to access the IBM DB2 Family forums

All DB2 products are supported through these forums.

To find out about the IBM Professional Certification Program for DB2 Universal Database, go to http://www.software.ibm.com/data/db2/db2tech/db2cert.html

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-2858-00